2.1.63. SQL Exercise

02:51

The customer management system at your organization requires an update to store additional information about customers. To achieve this, the **customers** table in the database needs to include a new column to store the age of each customer. This new column will help the organization better segment and analyze customer demographics.

Write an SQL query to add a new column named **age** of data type INTEGER to the existing customers table. Ensure that the schema modification does not alter or delete any existing data in the table.

Table Schema:

Column Name	Data Type	Constraints
customer_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
email	VARCHAR(255)	NOT NULL
address	TEXT	-
phone_number	VARCHAR(50)	-

Data Snaptshot for customers table:

customer_id	name	email	address	phone_number
1	John Doe	johndoe@email.com	123 Apple St.	123-456-7890
2	Jane Smith	janesmith@email.com	456 Orange Ave.	234-567-8901
3	Bob Johnson	bobj@email.com	789 Banana Rd.	345-678-9012
4	Alice Brown	alice@email.com	321 Cherry Blvd.	456-789-0123

Explorer

user.sql

```
1  --Type SQL here
2  ALTER TABLE customers ADD COLUMN age INT;
```

2.1.64. ALTER TABLE Command - Delete a column from table

00:43

Deleting a column from a table in PostgreSQL is a crucial database management operation. This task is often needed when a specific piece of data becomes irrelevant, or the database schema is being streamlined. While it's a powerful tool for database optimization, it should be used cautiously, as it results in the permanent removal of the column and its data.

Command, Syntax, and Examples

Deleting a column from a table
Syntax:

```
ALTER TABLE table_name DROP COLUMN column_name;
```

Example:
Removing the category column from the products table:

```
ALTER TABLE products DROP COLUMN category;
```

This command deletes the category column and all its data from the products table.

Task:
In the CodeKraft database, there's a need to remove the phone_number column from the existing customers table, as it's no longer required for the business process.

Question:
Which SQL statement(s) correctly delete the phone_number column from the existing customers table in the CodeKraft database? (Select all that apply)

- [x] `ALTER TABLE customers DROP COLUMN phone_number;`
- [ ] `DELETE COLUMN phone_number FROM customers;`
- [ ] `REMOVE COLUMN phone_number FROM TABLE customers;`
- [ ] `ALTER TABLE customers DELETE COLUMN phone_number;`

2.1.65. SQL Exercise

01:26

The inventory management system in your organization is undergoing optimization to simplify its database structure. As part of this update, the **description** column in the **products** table is no longer required for business operations. To streamline the database, this column must be removed.

Write an SQL query to remove the **description** column from the **products** table in the inventory database. Ensure that this operation does not affect the integrity or data in the remaining columns.

Original Table Schema: products

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(100)	-
price	NUMERIC(10,2)	NOT NULL
description	TEXT	-
stock_quantity	INTEGER	-

Data in products Table:

product_id	name	category	price	description	stock_quantity
1	Smartphone	Electronics	999.99	High-end smartphone	50
2	T-shirt	Clothing	19.99	Cotton t-shirt	200
3	Backpack	Accessories	49.99	Durable backpack	100

Explorer

user.sql

```
1  ALTER TABLE products DROP COLUMN description;
```

2.1.66. ALTER TABLE Command - Change the data type of a column

00:56

Changing the data type of a column in PostgreSQL is an essential aspect of database management, particularly when the nature of the data in that column evolves or was initially misconfigured. This operation allows for the modification of the column's data type to better reflect the kind of data stored, ensuring data integrity and optimal database performance.

Command, Syntax, and Examples

Changing the Data Type of a Column
Syntax:

```
ALTER TABLE table_name ALTER COLUMN column_name TYPE new_data_type;
```

Example:
Changing the data type of the price column in the products table from NUMERIC to FLOAT:

```
ALTER TABLE products ALTER COLUMN price TYPE FLOAT;
```

This command alters the price column in the products table to have a data type of FLOAT instead of its previous NUMERIC type.

Task:
In the CodeKraft database, the customer_reviews table has a column rating initially set as VARCHAR. There's a need to change this column's data type to INT as it should store numerical values.

Question:
Which SQL statement(s) correctly change the data type of the rating column to INT in the customer_reviews table in the CodeKraft database? (Select all that apply)

- [ ] `ALTER TABLE customer_reviews MODIFY COLUMN rating INT;`
- [x] `ALTER TABLE customer_reviews ALTER COLUMN rating TYPE INT;`
- [ ] `UPDATE TABLE customer_reviews CHANGE COLUMN rating rating INT;`
- [ ] `ALTER TABLE customer_reviews CHANGE rating rating INT;`

2.1.67. SQL Exercise

02:15

The E-Shop Logistics team is revamping its database structure to better align with financial regulations and improve operational clarity. Currently, the database includes a table named **orders**, which holds customer order information. However, this table needs a more descriptive name. Additionally, the column **total_amount**, which stores the monetary value of orders, is defined with a data type that is insufficient for handling larger amounts accurately.

You are required to perform the following queries:

- **Query 1:** Rename the **orders** table to **client_purchases** to make the table's purpose more explicit.
- **Query 2:** Update the **total_amount** column's data type from **NUMERIC(10, 2)** to **DECIMAL(12, 2)** to comply with financial standards for high-value transactions.

Table Schema:

Column Name	Data Type	Constraints
order_id	SERIAL	PRIMARY KEY
customer_id	INTEGER	-
order_date	DATE	NOT NULL
total_amount	NUMERIC(10,2)	-
status	VARCHAR(100)	-

Sample Data:

order_id	customer_id	order_date	total_amount	status
1	1	2023-12-01	1059.97	Delivered
2	2	2023-12-02	79.96	Shipped
3	3	2023-12-03	58.98	Processing

Explorer

user.sql

```sql
--Query 1: Rename the orders table to client_purchases
ALTER TABLE orders RENAME TO client_purchases;

--Query 2: Update the total_amount column's data type from NUMERIC
DECIMAL(12,2)
ALTER TABLE client_purchases ALTER COLUMN total_amount TYPE DECIMAL
```

2.1.68. ALTER TABLE Command - Renaming a Columns

00:39

Changing a column name in PostgreSQL is a key operation in database management, especially when refining or correcting your database schema. Renaming a column can be necessary for various reasons, such as making the column name more descriptive, aligning with new naming conventions, or correcting a typo. This action requires caution as it might impact existing queries and applications relying on the original column name.

Command, Syntax, and Examples

Changing a Column Name
Syntax:

```
ALTER TABLE table_name RENAME COLUMN current_column_name TO new_column_name;
```

Example:
Renaming the cust_email column to customer_email in the customers table:

```
ALTER TABLE customers RENAME COLUMN cust_email TO customer_email;
```

This command changes the name of the cust_email column in the customers table to customer_email.

Task:
In the CodeKraft database, the orders table has a column named order_date_time. It needs to be renamed to order_date for clarity and consistency.

Question:
Which SQL statement(s) correctly change the name of the order_date_time column to order_date in the orders table in the CodeKraft database? (Select all that apply)

- [ ] `UPDATE TABLE orders CHANGE COLUMN order_date_time TO order_date;`
- [x] `ALTER TABLE orders RENAME COLUMN order_date_time TO order_date;`
- [ ] `ALTER TABLE orders CHANGE order_date_time TO order_date;`
- [ ] `RENAME COLUMN order_date_time TO order_date IN TABLE orders;`

2.1.69. SQL Exercise

03:37

The CodeClear Product Review Team has decided to streamline the **product_reviews** table in their database for better performance and consistency. To achieve this, several schema updates need to be implemented.

You are required to perform the following modifications to the **product_reviews** table in the CodeClear database:

- **Query 1: Rename the table** from **product_reviews** to **reviews** to simplify the table name.
- **Query 2: Modify the data type** of the **rating** column from **INTEGER** to **SMALLINT** for more efficient storage, as ratings are always small values.
- **Query 3: Remove the comment** column as it is deemed unnecessary for the current application.
- **Query 4: Rename the product_id** column to **item_id** to align with the naming convention used across other tables.

Table Schema:

Column Name	Data Type	Constraints
review_id	SERIAL	PRIMARY KEY
product_id	INTEGER	-
customer_id	INTEGER	-
rating	INTEGER	-
comment	TEXT	-

Data for product_reviews:

review_id	product_id	customer_id	rating	comment
1	1	1	5	Excellent smartphone!
2	2	2	4	Comfortable t-shirt.
3	3	3	3	Decent backpack.

Explorer

user.sql

```sql
-- Rename the table from product_reviews to reviews
ALTER TABLE product_reviews RENAME TO reviews;

-- Modify the data type of the rating column from INTEGER to SMALLI
ALTER TABLE reviews ALTER COLUMN rating TYPE SMALLINT;

-- Remove the comment column
ALTER TABLE reviews DROP COLUMN comment;

-- Rename the product_id column to item_id
ALTER TABLE reviews RENAME COLUMN product_id TO item_id;


```

2.1.69. SQL Exercise

03:37

The CodeClear Product Review Team has decided to streamline the **product_reviews** table in their database for better performance and consistency. To achieve this, several schema updates need to be implemented.

You are required to perform the following modifications to the **product_reviews** table in the CodeClear database:

- **Query 1: Rename the table** from **product_reviews** to **reviews** to simplify the table name.
- **Query 2: Modify the data type** of the **rating** column from **INTEGER** to **SMALLINT** for more efficient storage, as ratings are always small values.
- **Query 3: Remove the comment** column as it is deemed unnecessary for the current application.
- **Query 4: Rename the product_id** column to **item_id** to align with the naming convention used across other tables.

Table Schema:

Column Name	Data Type	Constraints
review_id	SERIAL	PRIMARY KEY
product_id	INTEGER	-
customer_id	INTEGER	-
rating	INTEGER	-
comment	TEXT	-

Data for product_reviews:

review_id	product_id	customer_id	rating	comment
1	1	1	5	Excellent smartphone!
2	2	2	4	Comfortable t-shirt.
3	3	3	3	Decent backpack.

Explorer

user.sql

```sql
-- Rename the table from product_reviews to reviews
ALTER TABLE product_reviews RENAME TO reviews;

-- Modify the data type of the rating column from INTEGER to SMALLI
ALTER TABLE reviews ALTER COLUMN rating TYPE SMALLINT;

-- Remove the comment column
ALTER TABLE reviews DROP COLUMN comment;

-- Rename the product_id column to item_id
ALTER TABLE reviews RENAME COLUMN product_id TO item_id;


```

2.1.70. DROP TABLE command

00:05

Deleting a table in PostgreSQL is a significant operation that involves permanently removing a table and all of its data from the database. This action is typically performed when a table becomes obsolete or irrelevant. It's a critical task that should be approached with caution, as once a table is deleted, all the information stored in it is lost and cannot be recovered unless backups are available.

Command, Syntax, and Examples

Deleting a Table
Syntax:

```
DROP TABLE table_name;
```

Example:
Deleting the temporary_data table:

```
DROP TABLE temporary_data;
```

This command removes the temporary_data table from the database, along with all its data.

Task:
In the CodeKraft database, there is a table named old_customer_records that is no longer in use and needs to be deleted.

Question:
Which SQL statement(s) correctly delete the old_customer_records table in the CodeKraft database? (Select all that apply)

- [ ] `DELETE TABLE old_customer_records;`
- [ ] `REMOVE TABLE old_customer_records;`
- [x] `DROP TABLE old_customer_records;`
- [ ] `ALTER TABLE old_customer_records DROP;`

Home Learn Anywhere

12411693.st@lpu.in Support

2.1.71. SQL Exercise

00:47

You are managing a database for a retail company, and due to changes in the business model, the company no longer needs to store customer information in the database. You have been assigned the task of removing the **customers** table from the database.

Write the SQL statement to drop the **customers** table from your PostgreSQL database.

Expected Output:
If you write and execute the correct command, the table will be deleted and the PostgreSQL typically displays a message like:

```
DROP TABLE
```

Explorer

user.sql

```
1  DROP TABLE customers;
```

2.1.72. TRUNCATE TABLE Command

00:50

Deleting all rows from a table while retaining the table structure in PostgreSQL is a common operation, often used in data refresh scenarios, testing, or clearing out data without removing the table itself. This action is achieved using the TRUNCATE command, which quickly removes all records from a table but leaves the table structure and its definitions, such as columns and constraints, intact.

Command, Syntax, and Examples

Deleting All Rows While Keeping Table Structure

Using TRUNCATE
Syntax:

```
TRUNCATE TABLE table_name;
```

Example:
Removing all data from the customer_logs table:

```
TRUNCATE TABLE customer_logs;
```

This command deletes all rows from the customer_logs table but keeps the table itself and its schema.

Using DELETE
Syntax:

```
DELETE FROM table_name;
```

Example:

```
DELETE TABLE customer_logs;
```

Task:
In the CodeKraft database, you need to delete all entries in the session_data table to clear out old user session records, but the table structure must remain for future data.

Question:
Which SQL statements correctly delete all rows from the session_data table while retaining its structure in the

- [x] `DELETE FROM session_data;`
- [ ] `DROP TABLE session_data;`
- [x] `TRUNCATE TABLE session_data;`
- [ ] `REMOVE ALL FROM session_data;`

2.1.73. SQL Exercise

02:02

You are working as a database administrator for a global retail company. The company is currently running a promotional campaign, and you need to refresh the customer data in the system. However, you must ensure that the structure of the customers table remains intact, and only the data is cleared. This action is part of the data preparation process before importing updated customer data.

Your task is to empty all the current records from the customers table while keeping the table structure intact, so that it can later be populated with fresh data.

Table Schema for customers:

Column Name	Data Type	Constraints
customer_id	SERIAL	PRIMARY KEY
name	VARCHAR (255)	NOT NULL
age	INTEGER	-
nationality	VARCHAR (100)	-
address	TEXT	-

Expected Output:
After performing the operation, the customers table will become empty, but its structure (columns and data types) will remain unchanged. No customer data should be present in the table after the operation. And the PostgreSQL typically displays a message like:

```
TRUNCATE TABLE
```

Explorer

user.sql

```
1  TRUNCATE TABLE customers
```

2.1.74. TRUNCATE ... RESTART IDENTITY Command

02:34

In PostgreSQL, when managing tables such as customer records or order details, it's sometimes necessary to delete all data while resetting auto-increment (identity) columns. This is particularly useful in scenarios like refreshing a test database or reinitializing data. The TRUNCATE ... RESTART IDENTITY command accomplishes this by removing all rows and resetting auto-increment counters.

Command, Syntax, and Examples

Deleting All Rows and Resetting Identity Columns
Syntax:

```
TRUNCATE TABLE table_name RESTART IDENTITY;
```

Example:
Clearing all data and resetting auto-increment in the order_details table:

```
TRUNCATE TABLE order_details RESTART IDENTITY;
```

This command will remove all rows from the order_details table and reset the auto-increment counter for the identity column.

Task:
In CodeKraft's database, there's a requirement to clear all data from the product_reviews table and reset the sequence of its primary key (assuming it's an auto-increment field) for a new set of product review data.

Question:
Which SQL statements correctly delete all rows from the product_reviews table and reset the auto-increment sequence of the identity column in the CodeKraft database? (Multiple answers possible)

- [x]
```
TRUNCATE TABLE product_reviews RESTART IDENTITY;
```

- [x]
```
DELETE FROM product_reviews;
ALTER SEQUENCE product_reviews_review_id_seq RESTART WITH 1;
```

- [ ]
```
TRUNCATE TABLE product_reviews;
```

- [ ]
```
DROP TABLE product_reviews;
CREATE TABLE product_reviews (...);
```

2.1.75. SQL Exercise

03:23

You are a database administrator for a large retail company that is preparing for a major inventory update. The **products** table contains information about all current products, including their IDs, names, categories, prices, descriptions, and stock quantities. Before importing the updated product data into the system, you need to clear the existing records from the **products** table while ensuring that the auto-increment value for the **product_id** column resets to its initial value. This will allow the new products to be added smoothly, starting from the first product ID.

Your goal is to execute the necessary operation to clear all the records from the **products** table and reset the auto-increment value for **product_id**.

Table Schema:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(100)	-
price	NUMERIC(10,2)	NOT NULL
description	TEXT	-
stock_quantity	INTEGER	-

Data for products table:

product_id	name	category	price	description	stock_quantity
1	Smartphone	Electronics	999.99	High-end smartphone	50
2	T-shirt	Clothing	19.99	Cotton t-shirt	200
				Durable	

Explorer

user.sql

```
1  DELETE FROM products;
2  ALTER SEQUENCE products_product_id_seq RESTART WITH 1;
```

2.1.76. Including comments in PostgreSQL

01:02

In PostgreSQL, comments are used to provide explanations or documentation within the code itself. PostgreSQL supports two types of comments: single-line comments and multi-line comments.

Single-Line Comments:
Single-line comments start with -- and extend until the end of the line. These comments are used for providing explanations or annotations on a single line of code.
Example:

```
-- This is a single-line comment
SELECT * FROM employees; -- This comment follows a statement
```

Multi-Line Comments:
Multi-line comments are enclosed within /* and */. They can span multiple lines and are useful for providing detailed explanations or for commenting out blocks of code temporarily.
Example:

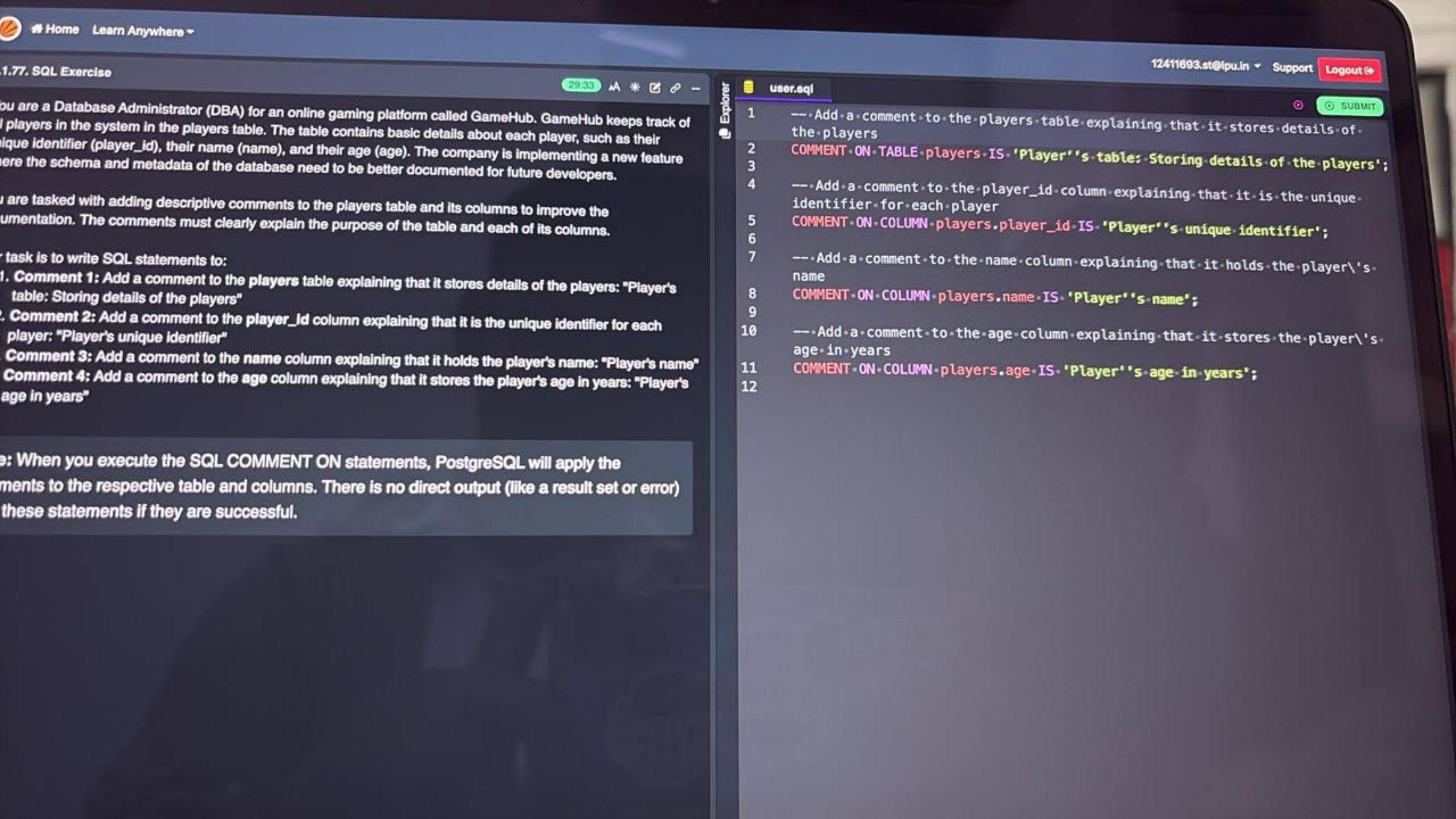
```
/* This is a
multi-line comment */
SELECT * FROM departments;
```

Note: In PostgreSQL, you cannot nest multi-line comments. This means you cannot have one multi-line comment inside another.

Using Comments Effectively:
Comments should be used effectively to document your SQL code, especially for complex queries or stored procedures. Here are some tips on using comments effectively:

- **Explain Complex Logic:** Use comments to explain complex logic or calculations in your queries, making it easier for others to understand your code.
- **Document Changes:** If you make changes to your code, update the comments accordingly to reflect the current logic.
- **Temporary Commenting:** When debugging or testing, you can comment out parts of your code temporarily to isolate issues without deleting the code.
- **Version Control:** In collaborative projects, comments can help team members understand the purpose of specific code sections and changes made over time.
- **Use Descriptive Comments:** Write descriptive comments that provide context and explain the "why" behind

- [ ] Single-line comments in PostgreSQL start with /*.
- [ ] Multi-line comments in PostgreSQL start with --.
- [x] Multi-line comments in PostgreSQL are enclosed within /* and */.
- [x] Nesting of multi-line comments is not supported in PostgreSQL.


Home
Learn Anywhere
1.77. SQL Exercise
29:33
12411693.st@lpu.in
Support
Logout
bu are a Database Administrator (DBA) for an online gaming platform called GameHub. GameHub keeps track of l players in the system in the players table. The table contains basic details about each player, such as their ique identifier (player_id), their name (name), and their age (age). The company is implementing a new feature ere the schema and metadata of the database need to be better documented for future developers.
u are tasked with adding descriptive comments to the players table and its columns to improve the umentation. The comments must clearly explain the purpose of the table and each of its columns.
task is to write SQL statements to:
1. Comment 1: Add a comment to the players table explaining that it stores details of the players: "Player's table: Storing details of the players"
2. Comment 2: Add a comment to the player_id column explaining that it is the unique identifier for each player: "Player's unique identifier"
3. Comment 3: Add a comment to the name column explaining that it holds the player's name: "Player's name"
4. Comment 4: Add a comment to the age column explaining that it stores the player's age in years: "Player's age in years"
e: When you execute the SQL COMMENT ON statements, PostgreSQL will apply the ments to the respective table and columns. There is no direct output (like a result set or error) these statements if they are successful.
Explorer
user.sql
SUBMIT
1 --·Add·a·comment·to·the·players·table·explaining·that·it·stores·details·of·the·players
2 COMMENT·ON·TABLE·players·IS·'Player''s·table:·Storing·details·of·the·players';
3
4 --·Add·a·comment·to·the·player_id·column·explaining·that·it·is·the·unique·identifier·for·each·player
5 COMMENT·ON·COLUMN·players.player_id·IS·'Player''s·unique·identifier';
6
7 --·Add·a·comment·to·the·name·column·explaining·that·it·holds·the·player\'s·name
8 COMMENT·ON·COLUMN·players.name·IS·'Player''s·name';
9
10 --·Add·a·comment·to·the·age·column·explaining·that·it·stores·the·player\'s·age·in·years
11 COMMENT·ON·COLUMN·players.age·IS·'Player''s·age·in·years';
12

You are working on a flight reservation system. The system needs to store essential flight and passenger details for flight management and bookings. You are tasked with creating a table to record flight details and passenger information. The table should track information like flight number, passenger name, origin, destination, and the class of service.

You will also need to alter the table by adding an additional column for seat_number, and then truncate all data when needed.

flight_reservations Table Schema:

Column Name	Data Type	Constraints	Description
flight_no	INT	PRIMARY KEY	A unique identifier for each flight.
passenger_name	VARCHAR(50)	NOT NULL	The name of the passenger.
origin	VARCHAR(50)	-	The origin city of the flight.
destination	VARCHAR(50)	-	The destination city of the flight.
class	VARCHAR(50)	-	The class of service (e.g., Economy, Business, First Class).

Flight Reservation System Requirements:

- **Step 1:** Create a table for storing flight reservation details as mentioned in the above schema
- **Step 2:** Alter the table to include a new column `seat_no` with VARCHAR(10) Data Type
- **Step 3:** Truncate all data from the table

Output:

- After creating the table and altering it, the table will have a new column `seat_num` (VARCHAR(10)) along with the above mentioned columns.
- After truncating the table, all data will be removed, leaving the table empty but still intact. The table schema remains the same, so it can be reused.

Table flight_reservations need to be created as per below schema

user.sql

SUBMIT

```sql
-- CREATE TABLE for storing flight reservation details
CREATE TABLE flight_reservations(
    flight_no INT PRIMARY KEY,
    passenger_name VARCHAR(50) NOT NULL,
    origin VARCHAR(50),
    destination VARCHAR(50),
    class VARCHAR(50)
);

....
-- Alter the table to include a new column
ALTER TABLE flight_reservations ADD COLUMN seat_no VARCHAR(10);

-- Insert values to table
INSERT INTO flight_reservations (flight_no, passenger_name, origin, destination, class, seat_no)
VALUES
    (104, 'Michael Brown', 'Boston', 'Denver', 'First', '1A'),
    (105, 'Emma Wilson', 'Atlanta', 'Dallas', 'Business', '2B'),
    (106, 'Olivia Davis', 'Seattle', 'Chicago', 'Economy', '3C');

-- Truncate all data from the table
TRUNCATE TABLE flight_reservations;
```

Terminal Test cases

2.2.2. DDL - ALTER - Library System Management

02:27

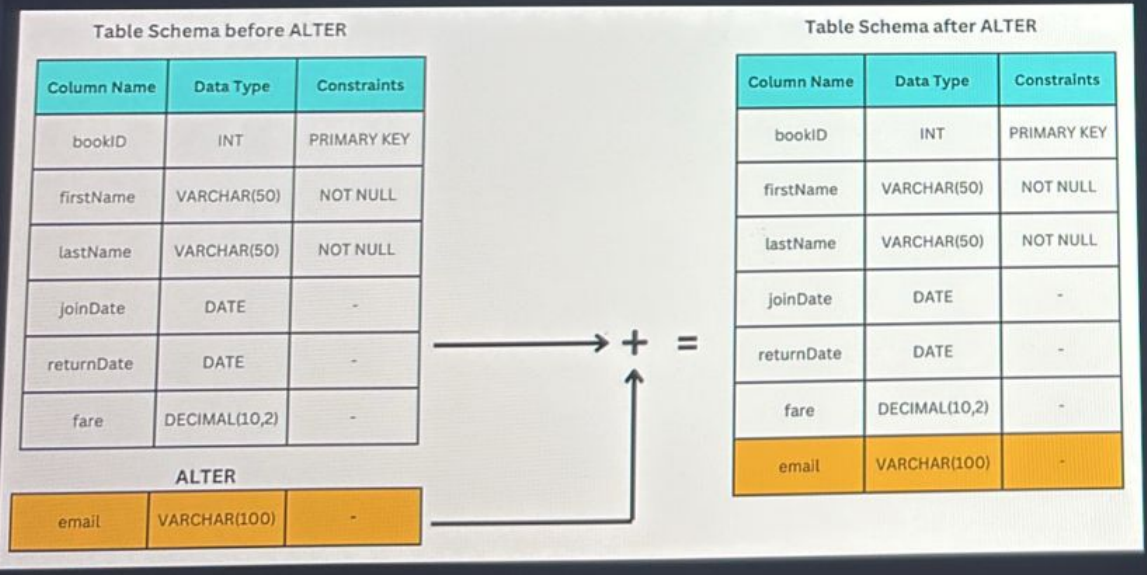
You are enhancing a Library Management System and need to add functionality to contact library members via email. To achieve this, you must update the **library_members** table by adding a new column named **email**. This column will store the email addresses of members and will help the library staff communicate with members about book due dates, events, and notifications.

Requirements:

- The column name should be `email`.
- The column data type should be `VARCHAR(100)`.
- Ensure the column can store up to 100 characters.
- The column should allow NULL values, as not all members may have an email address initially.

Write an SQL statement to implement this requirement.

Note: You are not required to create the table, The table is already available in the database.



Output: The **email** column is added to the `library_members` table.

Explorer

index.sql

```sql
-- Add the email column
ALTER TABLE library_members ADD COLUMN email VARCHAR(100);
```

2.2.3. DDL - Police Department Information Management System

00:45

You are working for the Police Department's Data Management System. Due to a sudden network breach, unauthorized access was gained to the police department's internal records, and sensitive officer information was compromised. As a result, all the data in the `police_officers` table must be removed immediately to prevent further misuse. However, the table structure must remain intact so that new and verified data can be inserted after the security breach has been resolved.

Task:
Your task is to remove all the records from the `police_officers` table as quickly as possible, but keep the table structure intact for future secure data entry.

Table Schema for police_officers:

Column Name	Data Type	Constraints
officerID	INT	PRIMARY KEY
firstName	VARCHAR(50)	NOT NULL
lastName	VARCHAR(50)	NOT NULL
rank	VARCHAR(50)	-
salary	DECIMAL(10,2)	-
dateOfJoining	DATE	-

Data Snapshot:

officerID	firstName	lastName	rank	salary	dateOfJoining
1	John	Doe	Sergeant	50000.00	2010-05-01
2	Jane	Smith	Lieutenant	60000.00	2015-08-10
3	Mark	Johnson	Captain	75000.00	2012-11-23

Explorer

index.sql

```sql
-- Remove all records from the table
TRUNCATE TABLE police_officers;

```

2.2.4. DDL - Weather Monitoring System

01:33

You are working on a Weather Monitoring System that collects real-time weather data from various regions. Recently, the system underwent a major overhaul, and all data from the old `weather_data` table has been successfully migrated to a new structure. As part of the cleanup process, the outdated `weather_data` table must be completely removed from the database to avoid any conflicts with the new design.

Requirements:

- The table name is `weather_data`.
- Ensure the table is dropped from the database without errors.

Write an SQL statement to drop the `weather_data` table from the database.

Expected Output:
The `weather_data` table will be permanently removed from the database.

Note: You are not required to create the table, The table is already available in the database.

Explorer

index.sql

```
1  DROP TABLE weather_data;
```

2.2.5. DDL - Employee Management System

11:06

You are working for a rapidly growing company called CodeTantra, which is expanding its operations. The HR department requires a robust employee management system to maintain detailed records of all employees. This system must store essential employee information, including their personal details, salaries, and hire dates. To ensure accurate record-keeping, you need to create a table named `employees` that adheres to the schema provided below.

`employees` table schema:

Column Name	Data Type	Constraints
employeeID	INT	PRIMARY KEY
firstName	VARCHAR(50)	NOT NULL
lastName	VARCHAR(50)	NOT NULL
birthDate	DATE	-
hireDate	DATE	-
salary	DECIMAL(10, 2)	CHECK (salary >= 0)

Requirements:

- The table name must be **employees.**
- The **employeeID** column will serve as the primary key to uniquely identify each employee.
- The **firstName** and **lastName** columns cannot be null, ensuring that every employee's name is recorded.
- The **salary** column must include a check constraint to ensure only non-negative salary values are allowed.
- Other columns like **birthDate** and **hireDate** are optional and can accept **NULL** values.

Write a SQL query to create the **employees** table with the schema and constraints described above.

Expected Outcome:

The employees table will be successfully created in the database.

- As no data is inserted or selected, running the query will not return any output.

Explorer

index.sql

```sql
CREATE TABLE employees(
    employeeID INT PRIMARY KEY,
    firstName VARCHAR(50) NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    birthDate DATE,
    hireDate DATE,
    salary DECIMAL(10,2) CHECK (salary>=0)
    );

```

2.2.6. DDL - Adding New Column

01:14

You are working on an employee management system for a large corporation. Recently, the company introduced a new department called "Innovation". To track which employees belong to this new department or any other department, you need to modify the existing `employees` table by adding a new column named `department`.

Requirements:

- The **employees** table already exists in the database.
- Add a new column named **department** to the table.
- The column **department** should have the data type **VARCHAR(100)**.
- This column will store the department name for each employee.
- The column can accept **NULL** values initially, as not all employees may be assigned to a department at the time of addition.

employees table schema:

Column Name	Data Type	Constraints
employeeID	INT	PRIMARY KEY
firstName	VARCHAR(50)	NOT NULL
lastName	VARCHAR(50)	NOT NULL
birthDate	DATE	-
hireDate	DATE	-
salary	DECIMAL(10,2)	CHECK (salary >= 0)

Write a SQL query to add the **department** column to the **employees** table.

Expected Output:

- The department column will be added to the employees table successfully.
- The query will execute without returning any result, as it only modifies the table structure.

Explorer

index.sql

```sql
-- Add the new column to the table
ALTER TABLE employees ADD COLUMN department VARCHAR(100);

```

2.2.7. Modifying Table Structures and Data: Alterations in SQL - 2

01:56

You are working with a table named emp in the database, which stores employee information. Perform the following tasks to update the table structure as required:

emp Table Schema (Before Modification):

Column Name	Data Type	Constraints
employeeID	INT	PRIMARY KEY
employeeName	VARCHAR(100)	-
salary	DECIMAL(10, 2)	-
depttname	VARCHAR(100)	-

Tasks 1: Add a New Column

Add a new column named **designation** to the **emp** table.

- **Column Name:** designation
- **Data Type:** VARCHAR(100)

Expected Outcome: The new column designation will be successfully added to the emp table.

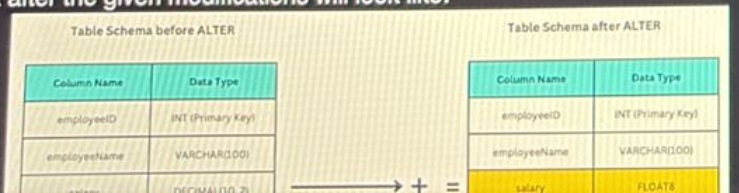
Task 2: Change the Data Type of salary

Change the data type of the salary column from DECIMAL(10, 2) to FLOAT8.

- **Column Name:** salary
- **New Data Type:** FLOAT8

Expected Outcome: The data type of the salary column will be updated to FLOAT8, allowing it to store floating-point values.

The emp table schema after the given modifications will look like:



Explorer

index.sql

```
1  --·Add·the·designation·Column
2  ALTER·TABLE·emp·ADD·COLUMN·designation·VARCHAR(100);
3
4
5  --·Change·the·Data·Type·of·salary
6  ALTER·TABLE·emp·ALTER·COLUMN·salary·TYPE·FLOAT8;
7
8
9
```

2.2.8. DDL - DROP - 1

00:30

You are working as a database administrator in a company, and you have been instructed to remove the `employees` table from the database. This table currently contains all employee information, and due to a change in the company's requirements, the table is no longer needed. Dropping the table will permanently delete the table along with all the data it holds.

Instructions:

- The employees table already exists in the database.
- Your task is to completely remove this table from the database.
- This action will permanently delete the table and all of its data, so it cannot be recovered unless a backup is available.

Write an SQL statement to drop the **employees** table from the database.

Expected Outcome:

- The **employees** table will be successfully removed from the database.
- All data within the table will be permanently deleted and cannot be recovered unless a backup is available.

Note: You are not required to create the table, The table is already available in the database.

Explorer

index.sql

```sql
-- Remove the table
DROP TABLE employees;

```

2.2.9. DDL - TRUNCATE - 1

00:28

You are a database administrator at a rapidly growing tech startup. The company's HR department has been maintaining detailed employee data in the `employees` table. However, due to a restructuring phase, the HR team needs to **temporarily remove all the records** from the table. The table's structure must remain intact so that new records can be added after the restructuring is complete.

The HR team has a tight deadline and needs this task to be completed without affecting any future operations that depend on the table structure, such as employee onboarding and payroll systems.

Requirements:

- **Objective:** Remove all records from the **employees** table, but keep the table's structure (columns, schema, constraints) intact.
- **Outcome:** The table must be cleared of data, but the column structure must remain unchanged so that new data can be inserted in the future.

`employees` table Schema:

Column Name	Data Type	Constraints
employeeID	INT	Primary Key
firstName	VARCHAR(50)	NOT NULL
lastName	VARCHAR(50)	NOT NULL
birthDate	DATE	-
hireDate	DATE	-
salary	DECIMAL(10,2)	CHECK (salary >= 0)

Data Snapshot:

employeeID	firstName	lastName	birthDate	hireDate	salary
			1980-01-15	2005-06-01	50000.00

Explorer

index.sql

```sql
-- Remove all records from the employees table
TRUNCATE TABLE employees;

```

2.2.10. DDL - 1

02:57

As the database administrator at a prestigious university, you are tasked with preparing the database for the new semester. The university needs to track student performance across various batches, and the system must store detailed information about each student, including their batch number, name, branch, and marks in three subjects.

The registrar needs to:

- **Create a table** to store student details, including batch number, name, branch, and marks for the first two subjects.
- **Add a new column** for marks in the third subject, as the curriculum now requires tracking marks for three subjects per student.
- **Clear any existing data** in the table before entering fresh records for the new batch of students.

Task 1: Create the student table with the following columns:

- batch_no (type: INT)
- name (type: VARCHAR(15))
- branch (type: VARCHAR(15))
- mark1 (type: INT)
- mark2 (type: INT)

Task 2: Alter the table to add a new column **mark3** (type: INT) to store the marks of the third subject.

Task 3: Truncate the table to remove all existing data, ensuring that the table is ready to store fresh records for the new batch.

Column Description:

Column Name	Description
batch_no	Indicates the batch number which is the type of integer
name	Indicates the student name which is the type of varchar having a size of 15 characters
branch	Indicates the branch of the student which is the type of varchar having a size of 15 characters
	Indicates the marks in the first subject which is the type of

Explorer

user.sql

```sql
-- Create the student table
CREATE TABLE student(
    batch_no INT,
    name VARCHAR(15),
    branch VARCHAR(15),
    mark1 INT,
    mark2 INT
);

-- Add the mark3 column
ALTER TABLE student ADD COLUMN mark3 INT;

-- Insert data
INSERT INTO student (batch_no, name, branch, mark1, mark2, mark
VALUES
    (101, 'Alice', 'CSE', 85, 90, 88),
    (102, 'Bob', 'ECE', 78, 82, 80),
    (103, 'Charlie', 'ME', 92, 88, 85);
....
-- Remove all existing data
TRUNCATE TABLE student;

```

2.1.2. Introduction to Data Definition Language (DDL) 06:05

Data Definition Language (DDL)

Overview

Data Definition Language (DDL) consists of SQL commands used to define the database schema in PostgreSQL. It deals with the descriptions of the database schema and is used to create, modify, and manage the structure of database objects.

DDL commands in PostgreSQL are used to create, modify, and delete database structures but not the data itself. These commands are generally used by database administrators or developers during the setup and modification phases of the database lifecycle, rather than by end-users who interact with the database through applications.

List of **DDL** Commands in **PostgreSQL**:

1. CREATE: Used to create a new database or its objects, such as tables, indexes, functions, views, stored procedures, and triggers.

Syntax:

```
CREATE TABLE table_name (...);
```

2. DROP: Used to delete objects from the database. In PostgreSQL, **DROP** can remove databases, tables, and other database objects.

Syntax:

```
DROP TABLE table_name;
```

3. ALTER: Used to modify the structure of existing database objects. In PostgreSQL, **ALTER** can be used to change various aspects of database objects, like adding a column to a table or changing a column's data type.

Syntax:

```
ALTER TABLE table_name ADD COLUMN new_column_name data_type;
```

4. TRUNCATE: Used to remove all records from a table quickly, without affecting the table's structure. This command also resets any auto-increment counters.

Syntax:

- [ ] The CREATE command in PostgreSQL is used to add new rows to an existing table.
- [ ] The DROP command in PostgreSQL is used to modify an existing column in a table.
- [x] The ALTER command in PostgreSQL is used to change the structure of an existing database objec adding a new column to a table.
- [ ] The TRUNCATE command in PostgreSQL is primarily used for updating data in existing table row

2.1.3. SQL data types

00:10

PostgreSQL supports a wide range of data types, allowing it to store different kinds of information efficiently. Understanding these data types is crucial for database design and data manipulation.

Main Data Types in PostgreSQL

Numeric Types

- **Integer Types:**
  a. **SMALLINT**: Stores 2-byte integers, ranging from `-32,768` to `32,767`. Ideal for small quantities, like a count of items that won't exceed the limit.
  b. **INTEGER**: Stores 4-byte integers, ranging from `-2,147,483,648` to `2,147,483,647`. Commonly used for quantities in a larger range, such as population counts.
  c. **BIGINT**: Stores 8-byte integers, ranging from `-9,223,372,036,854,775,808` to `9,223,372,036,854,775,807`. Suitable for very large numbers, like astronomical calculations.
- **Decimal Types:**
  a. **DECIMAL** and **NUMERIC**: Both types store exact numeric values. Used in financial calculations where precision is crucial, like storing prices or interest rates. The precision and scale can be specified.
- **Floating-Point Types:**
  a. **REAL**: 4-byte floating-point number. Used for scientific calculations where precision is less critical.
  b. **DOUBLE PRECISION**: 8-byte floating-point number. Used when higher precision floating-point numbers are needed.

Character Types

- **CHAR(n) / CHARACTER(n)**: Fixed-length, right-padded with spaces. Useful for storing data with a fixed format, like country codes (e.g., 'US', 'UK').
- **VARCHAR(n) / CHARACTER VARYING(n)**: Variable-length strings with a specified limit. Ideal for names, email addresses, or other strings where the maximum length is known but may vary.
- **TEXT**: Unlimited length. Used for long-form text, such as descriptions, emails, or articles.

Date/Time Types

- **DATE**: Stores date (no time). Used for birth dates, event dates, etc.
- **TIME**: Stores time of day, with or without time zone. Useful for storing times of events.
- **TIMESTAMP**: Stores both date and time. Ideal for recording exact moments of transactions or events, like order timestamps in an e-commerce system.

Boolean Type

...(TRUE, FALSE, and NULL). Commonly used for flags, such as indicating

- [ ] SMALLINT in PostgreSQL can be used for storing the population of a country.
- [x] NUMERIC data type is especially useful in scenarios where rounding errors cannot be tolerated, like financial records.
- [x] TEXT data type is recommended for storing lengthy text content such as customer reviews or article conten
- [ ] A BOOLEAN data type can store three values: TRUE, FALSE, and UNKNOWN.
- [ ] TIMESTAMP without time zone is suitable for storing the exact time of global events, such as an internati webinar.

2.1.4. CREATE DATABASE Command

00:10

Creating a database in PostgreSQL is a fundamental task for any database administrator or developer. A database acts as a container for all your tables, views, and other relational data structures. It's the first step in setting up a new project or system within PostgreSQL and is crucial for organizing and managing data effectively.

Command, Syntax, and Examples

Creating a New Database
Syntax:

```
CREATE DATABASE database_name;
```

Example:
Creating a database named codekraft_db

```
CREATE DATABASE codekraft_db;
```

This command creates a new database named codekraft_db. This database can then be used to store tables like products, customers, and others.

Creating a Database with additional options
Syntax:

```
CREATE DATABASE database_name
WITH
OWNER = role_name
TEMPLATE = template
ENCODING = encoding_type
LC_COLLATE = collate
LC_CTYPE = ctype
TABLESPACE = tablespace_name
CONNECTION LIMIT = max_connections;
```

Where,
1. WITH: This clause is used to specify additional options for the database creation. It is optional but allows fine-tuning of the database settings
2. OWNER: Specifies the role (user) that will own the database and the role_name is the username of the database owner. By default, the database owner is the user executing the command.
3. TEMPLATE: Specifies the template database to be used for creating the new database.
3. ENCODING: Defines the character encoding for the database, which determines how text data is stored.

- [x]
```
CREATE DATABASE codekraft_shop
WITH
OWNER = codekraft_admin
ENCODING = 'UTF8';
```

- [ ]
```
CREATE DB codekraft_shop
OWNER codekraft_admin
SET ENCODING 'UTF8';
```

- [x]
```
CREATE DATABASE codekraft_shop
WITH
ENCODING = 'UTF8'
OWNER = codekraft_admin;
```

- [ ]
```
NEW DATABASE codekraft_shop
ENCODED 'UTF8'
OWNED BY codekraft_admin;
```

2.1.6. ALTER DATABASE Command

Renaming a database in PostgreSQL is a task that involves changing the name of an existing database. It's a straightforward operation but crucial, especially in cases where the purpose of the database evolves or when aligning with new naming conventions.

Renaming a database can help in maintaining clarity, consistency, and relevance in your database management system.

Command, Syntax, and Examples

Renaming a Database
Syntax:

```
ALTER DATABASE current_database_name RENAME TO new_database_name;
```

Example:
Renaming a database from old_db to new_db

```
ALTER DATABASE old_db RENAME TO new_db;
```

This command changes the name of the database from old_db to new_db.

Task:
In CodeKraft's PostgreSQL setup, there is a need to rename the existing database codekraft_initial to codekraft_main.

Question:
Which SQL statements correctly renames the codekraft_initial database to codekraft_main? (Multiple answers possible)

- [ ] `RENAME DATABASE codekraft_initial TO codekraft_main;`
- [ ] `ALTER DATABASE codekraft_initial CHANGE TO codekraft_main;`
- [x] `ALTER DATABASE codekraft_initial RENAME TO codekraft_main;`
- [ ] `UPDATE DATABASE codekraft_initial SET NAME = codekraft_main;`

2.1.7. SQL ExerciseAdapting to Organizational Changes in Database Naming

00:45

CodeTantra has recently undergone a rebranding effort and decided to reflect these changes across all its resources, including databases. As part of this initiative, the database previously named "codetantra", which stores critical platform data, must be renamed to "renamed_codetantra" to align with the new naming conventions.

Your task is to write an SQL query to rename the existing "codetantra" database to "renamed_codetantra".

Expected Output:
When you write and run the correct command, the mentioned database will be renamed (altered) to the new name and the PostgreSQL displays a message like:

```
ALTER DATABASE
```

Explorer

user.sql

```
1  -- Type SQL here
2  ALTER DATABASE codetantra RENAME to renamed_codetantra;
```

Home Learn Anywhere

12411693.st@lpu.in Support Logout

2.1.5. Creating a New Database in PostgreSQL

11:43

As a database administrator for CodeTantra, you are tasked with setting up a new database for managing the company's upcoming e-learning platform. The database will store all user-related data, course information, and transaction records.

Your task is to write an SQL query to create a new database named "codetantra" in PostgreSQL.

Expected Output:
When you write and run the correct command, the mentioned database will be created and the PostgreSQL displays a message like:

```
CREATE DATABASE
```

Explorer

user.sql

```
1  CREATE DATABASE codetantra;
2
```

2.1.9. SQL Exercise

00:31

You are working as a database administrator for a company that manages its online retail platform using PostgreSQL. The company has decided to discontinue an experimental database named "customer_orders" as it is no longer required.

Your task is to write a SQL query to permanently drop the "customer_orders" database from the PostgreSQL database management system used by CodeTantra.

Note:

- This operation is irreversible, and all the data in the database will be permanently lost.
- Double-check the database name before execution to avoid accidental deletion of a critical database.

Expected Output:

If the database name is correct, the query will successfully delete the database. Otherwise, it will return an error indicating the database does not exist.

Explorer

user.sql

```
1  -- Type SQL here
2  DROP DATABASE customer_orders;
```

2.1.10. CREATE TABLE Command

00:52

In PostgreSQL, creating tables involves establishing a database structure by defining a table with or without columns, which serves as a fundamental step in database design to organize and manage data effectively, whether as a placeholder for future development or with a fully specified schema.

Command, Syntax, and Examples

1. Creating an empty table with no columns
Syntax:

```
CREATE TABLE table_name ();
```

Example:
Creating an empty table named future_data without any columns:

```
CREATE TABLE future_data ();
```

This command creates a new table named future_data in the database. Initially, this table does not contain any columns.

2. Creating Tables with Columns
Syntax:

```
CREATE TABLE table_name (
    column1_name column1_datatype,
    column2_name column2_datatype,
    ...
);
```

Example:
Creating a customers table with several basic columns:

```
CREATE TABLE customers (
  customer_id INTEGER,
  name VARCHAR(255),
  email VARCHAR(255),
  address TEXT,
  phone_number VARCHAR(15)
);
```

- ● `CREATE TABLE placeholder_table ();`
- ○ `CREATE TABLE placeholder_table;`
- ○ `CREATE EMPTY TABLE placeholder_table;`
- ○ `CREATE TABLE placeholder_table (id SERIAL PRIMARY KEY);`

2.1.8. DROP DATABASE Command

00:06

Deleting a database in PostgreSQL is an irreversible operation that completely removes a database and all its data. It is a crucial task for database management, particularly when a database becomes redundant or for data cleanup. However, due to its permanent nature, this action must be executed with caution.

Command, Syntax, and Examples

Deleting a Database
Syntax:

```
DROP DATABASE database_name;
```

Example:
Deleting a database named code_temp

```
DROP DATABASE code_temp;
```

Executing this command will permanently remove the code_temp database from PostgreSQL.

Task:
You need to delete a database named code_temp from the CodeTantra PostgreSQL setup, as it is no longer in use.

Question:
Which SQL statement(s) correctly deletes the code_temp database? (Multiple answers possible)

- [ ] `DELETE DATABASE code_temp;`
- [x] `DROP DATABASE code_temp;`
- [ ] `REMOVE DATABASE code_temp;`
- [ ] `ALTER DATABASE code_temp DROP;`

2.1.11. Preparing a Table for Customer Records in a Legacy System

02:03

You are part of the database team responsible for migrating customer data from an old system to a PostgreSQL database. The legacy system does not enforce primary keys or automatically generate unique IDs. However, the migration team needs a table named "customers" to store the data without any primary keys or serial data types, as the unique identifiers will be managed manually.

Your task is to write an SQL query to create the "customers" table in PostgreSQL, adhering to the following schema:

Table Schema:

Column Name	Description	Data Type
customer_id	A unique identifier for each customer.	INTEGER
first_name	The first name of the customer.	TEXT
last_name	The last name of the customer.	TEXT
date_of_birth	The date of birth of the customer.	DATE
email	The customer's email address.	TEXT
phone_number	The customer's phone number.	TEXT

Expected Output:

When you write and run the correct command, the mentioned table will be created and the PostgreSQL displays a message like:

```
CREATE TABLE
```

Explorer

user.sql

```
CREATE TABLE customers(
customer_id INTEGER,
first_name TEXT,
last_name TEXT,
date_of_birth DATE,
email TEXT,
phone_number TEXT
);
```

2.1.12. Primary Key

02:38

In PostgreSQL, a primary key is a column or a group of columns used to uniquely identify each row in a table. The primary key constraint ensures that no two rows have the same primary key value and that a primary key column cannot have NULL values. Defining a primary key is essential for maintaining the uniqueness and integrity of data within a table.

Command, Syntax, and Examples

Creating a Table with a single Primary Key column
Syntax:

```
CREATE TABLE table_name (
    primary_key_column_name data_type PRIMARY KEY,
    ...
);
```

or

```
CREATE TABLE table_name (
    primary_key_column_name data_type,
    PRIMARY KEY(primary_key_column_name)
    ...
);
```

Example:
Creating a products table with a primary key column:

```
CREATE TABLE products (
    product_id INTEGER PRIMARY KEY,
    name VARCHAR(255),
    price NUMERIC(10,2)
);
```

or

```
CREATE TABLE products (
    product_id INTEGER,
```

- [x]
```
CREATE TABLE customer_support_tickets (
    ticket_id INTEGER PRIMARY KEY
);
```

- [x]
```
CREATE TABLE customer_support_tickets (
    ticket_id INT PRIMARY KEY
);
```

- [x]
```
CREATE TABLE customer_support_tickets (
    ticket_id INT,
    PRIMARY KEY (ticket_id)
);
```

- [ ]
```
CREATE TABLE customer_support_tickets (
    id SERIAL,
    PRIMARY KEY (id)
);
```

2.1.13. SQL Exercise

08:23

An e-commerce platform wants to store reviews submitted by customers for various products in a table named product_reviews. The table should adhere to the following schema:

Table Schema:

Column Name	Description
review_id	Primary key, integer type, unique identifier for each review
product_id	Integer type, links to the specific product being reviewed.
customer_id	Integer type, identifies the customer providing the review.
rating	Integer type, denotes the rating given by the customer (range: 1 to 5).
comment	Text type, contains the customer's comments about the product.

Expected Output:

When you write and run the correct command, the mentioned table will be created and the PostgreSQL displays a message like:

```
CREATE TABLE
```

Explorer

user.sql

```
1   --Type SQL here
2   CREATE TABLE product_reviews(
3       review_id INT NOT NULL,
4       product_id INT NOT NULL,
5       customer_id INT NOT NULL,
6       rating INT,
7       comment TEXT,
8       PRIMARY KEY (review_id,product_id,customer_id)
9   );
```

2.1.14. Composite Key

00:44

In PostgreSQL, a composite primary key is defined using more than one column. It is used to uniquely identify each row in a table where a single column is not sufficient to ensure uniqueness. This type of primary key is crucial when the uniqueness of a record is identified by a combination of two or more columns.

Command, Syntax, and Examples

Creating a Table with multiple primary key columns
Syntax:

```
CREATE TABLE table_name (
    column1_name data_type,
    column2_name data_type,
    ...
    PRIMARY KEY (column1_name, column2_name, ...)
);
```

Example:
Creating a product_orders table with a composite primary key

```
CREATE TABLE product_orders (
    order_id INT,
    product_id INT,
    PRIMARY KEY (order_id, product_id)
);
```

Task:
In the CodeTantra updated database, you need to create a new table called **product_orders** to manage orders of products. This table should include columns for order_id and product_id, and both these columns together should form a composite primary key, as each product-order combination is unique.

Question:
Which SQL statements correctly create the product_orders table in the CodeTantra database with order_id and product_id as a composite primary key?

- [x]
```
CREATE TABLE product_orders (
    order_id INT,
    product_id INT,
    PRIMARY KEY (order_id, product_id)
);
```

- [ ]
```
CREATE TABLE product_orders (
    order_id INT PRIMARY KEY,
    product_id INT PRIMARY KEY
);
```

- [ ]
```
CREATE TABLE product_orders (
    order_id INT,
    product_id INT,
    PRIMARY KEY (order_id),
    PRIMARY KEY (product_id)
);
```

- [ ]
```
CREATE TABLE product_orders (
    order_id INT,
    product_id INT,
    PK (order_id, product_id)
);
```

2.1.15. NOT NULL Constraint

01:18

The **NOT NULL** constraint in PostgreSQL plays a pivotal role in ensuring data integrity. It is used in table definitions to prevent null values from being entered into a specific column. By enforcing the presence of meaningful data in certain fields, this constraint helps maintain the reliability and accuracy of the database.

It is particularly important in fields that represent critical information, such as identifiers, names, or other key attributes.

Command, Syntax, and Examples

Applying NOT NULL constraint while Table Creation

Syntax:

```
CREATE TABLE table_name (
    column_name data_type NOT NULL,
    ...
);
```

Example:

Creating the products table with mandatory name and price:

```
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    price NUMERIC(10,2) NOT NULL,
    category VARCHAR(100),
    description TEXT,
    stock_quantity INTEGER
);
```

In this example, every product must have a name and a price. The **NOT NULL** constraint ensures that these essential details are always provided.

Adding NOT NULL to an existing column

Syntax:

```
ALTER TABLE table_name ALTER COLUMN column_name SET NOT NULL;
```

```
CREATE TABLE customer_feedback (
  feedback_id INTEGER PRIMARY KEY,
  customer_id INTEGER NOT NULL,
  feedback_text TEXT NOT NULL
);
```

```
CREATE TABLE customer_feedback (
  feedback_id INTEGER PRIMARY KEY,
  customer_id INTEGER,
  feedback_text TEXT
);
```

```
CREATE TABLE customer_feedback (
  feedback_id INTEGER PRIMARY KEY,
  customer_id INTEGER REFERENCES customers(customer_id),
  feedback_text TEXT NOT NULL
);
```

```
CREATE TABLE customer_feedback (
  feedback_id INTEGER,
  customer_id INTEGER NOT NULL,
  feedback_text TEXT
);
```

2.1.16. SQL Exercise

04:14

Imagine you are developing a database for a new e-commerce platform named ShopSmart. The platform needs to store customer details securely and systematically. Your task is to design the customers table within the PostgreSQL database. This table will be used to store essential customer information, including their names, email addresses, physical addresses, and phone numbers.

Write an SQL statement to create the customers table according to the given schema.

Table Schema:

Column Name	Data Type	Constraints
customer_id	INTEGER	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
email	VARCHAR(255)	NOT NULL
address	TEXT	-
phone_number	VARCHAR(50)	-

Expected Output:

When you write and run the correct command, the mentioned table will be created and the PostgreSQL displays a message like:

```
CREATE TABLE
```

Explorer

user.sql

```sql
-- Type SQL here
CREATE TABLE customers(
    customer_id INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    address TEXT,
    phone_number VARCHAR(50)
);
```

2.1.17. SERIAL Column

In PostgreSQL, a SERIAL column is used for generating unique integer identifiers automatically. This is particularly useful for primary key columns, where each row requires a unique identifier. Using SERIAL simplifies the process of creating unique identifiers, as it automatically increments the value with each new row, starting from 1.

Command, Syntax, and Examples

Creating a table with a SERIAL column
Syntax:

```
CREATE TABLE table_name (
    serial_column_name SERIAL,
    ...
);
```

Example:
Creating the orders table with a SERIAL primary key:

```
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    total_amount NUMERIC(10,2)
);
```

In this example, order_id is a SERIAL column, which automatically assigns a unique ID to each order.

> **Note:** When the column is declared as SERIAL, data type need not be mentioned

Task:
In the CodeTantra database, create a new table called **customer_queries** to manage customer inquiries. The table should include a query_id column as a SERIAL type to automatically generate unique IDs for each query.

Question:

- [x]
```
CREATE TABLE customer_queries (
    query_id SERIAL PRIMARY KEY,
    customer_id INT,
    query_text TEXT,
    response TEXT
);
```

- [ ]
```
CREATE TABLE customer_queries (
    query_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    query_text TEXT,
    response TEXT
);
```

- [x]
```
CREATE TABLE customer_queries (
    query_id SERIAL,
    customer_id INT,
    query_text TEXT,
    response TEXT,
    PRIMARY KEY (query_id)
);
```

- [ ]
```
CREATE TABLE customer_queries (
    id SERIAL PRIMARY KEY,
    customer_id INT,
    query_text TEXT,
    response TEXT
);
```

12411693.st@lpu.in

2.1.18. SQL Exercise

02:50

You are developing the backend database for QuickCart, an online grocery delivery platform. One of the core features of the platform is tracking customer orders in real-time. To enable this, you need to create an **orders** table in the database that will store essential details about each order. This table will be used to track order history, manage customer interactions, and calculate business analytics such as total sales and delivery performance.

As part of the database setup for QuickCart, write an SQL statement to create the **orders** table according to the given schema.

Table Schema:

Column Name	Description
order_id	Unique identifier for each order, SERIAL, primary key.
customer_id	Customer's identifier, integer type.
order_date	Date of the order, DATE data type.
total_amount	Total monetary value of the order, numeric type.
status	Current status of the order, VARCHAR with maximum length of 20 characters.

Expected Output:

When you write and run the correct command, the mentioned table with given schema will be created and the PostgreSQL displays a message like:

```
CREATE TABLE
```

Explorer

user.sql

```
1  CREATE TABLE orders(
2      order_id SERIAL PRIMARY KEY,
3      customer_id INT,
4      order_date DATE,
5      total_amount NUMERIC,
6      status VARCHAR(20)
7  );
```

00:45

2.1.19. UNIQUE column

In PostgreSQL, the **UNIQUE** constraint ensures that all values in a column are distinct from one another. This constraint is applied to columns where each record must have a unique value, such as email addresses or usernames. It is crucial for maintaining data integrity and preventing duplicates in critical fields.

Command, Syntax, and Examples

Creating a Table with UNIQUE columns
Syntax:

```
CREATE TABLE table_name (
    column_name data_type UNIQUE,
    ...
);
```

or

```
CREATE TABLE table_name (
    column1_name data_type ,
    ...
    UNIQUE (column1_name)
);
```

Example:
Creating the customers table with a unique email column:

```
CREATE TABLE customers (
    customer_id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE,
    name VARCHAR(255)
);
```

or

```
CREATE TABLE customers (
    customer_id SERIAL PRIMARY KEY,
    email VARCHAR(255),
    name VARCHAR(255),
    UNIQUE (email)
);
```

- [x]
```
CREATE TABLE order_details (
    order_detail_id INT UNIQUE,
    order_id INT,
    product_id INT
);
```

- [ ]
```
CREATE TABLE order_details (
    order_detail_id INT,
    order_id INT,
    product_id INT
    NOT NULL (order_detail_id)
);
```

- [x]
```
CREATE TABLE order_details (
    order_detail_id INT,
    order_id INT,
    product_id INT,
    UNIQUE (order_detail_id)
);
```

- [ ]
```
CREATE TABLE order_details (
    detail_id INT NOT NULL,
    order_id INT,
    product_id INT,
);
```

Home Learn Anywhere

02:29

2.1.20. SQL Exercise

You are tasked with creating a product inventory table for a retail company. The company sells various products and needs to store information about each product, including its unique identifier, name, category, price, description, and stock quantity. The company also has a rule that no two products can belong to the same category with identical names, so the category should be unique.

Your task is to write the SQL statement to create this table **products** in PostgreSQL.

products Table Design:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	-
category	VARCHAR(100)	UNIQUE
price	NUMERIC(10,2)	-
description	TEXT	-
stock_quantity	INTEGER	-

Expected Output:
When you write and run the correct command, the mentioned table with given schema will be created and the PostgreSQL displays a message like:

```
CREATE TABLE
```

Explorer

user.sql

```
CREATE TABLE products(
    product_id SERIAL PRIMARY KEY,
    name VARCHAR(255),
    category VARCHAR(100) UNIQUE,
    price NUMERIC(10,2),
    description TEXT,
    stock_quantity INT
);
```

2.1.21. GENERATED ALWAYS AS IDENTITY Column

01:08

The GENERATED ALWAYS AS IDENTITY column in PostgreSQL is an advanced feature for automatically generating unique values for a column, typically used for primary keys. It provides more control than the traditional SERIAL datatype, adhering closely to the SQL standard.

Command, Syntax, and Examples

Creating a Table with a GENERATED ALWAYS AS IDENTITY Column
Syntax:

```
CREATE TABLE table_name (
    column_name data_type GENERATED ALWAYS AS IDENTITY,
    ...
);
```

Example:
Creating the inventory_logs table with an identity column:

```
CREATE TABLE inventory_logs (
    inventory_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    product_id INT,
    change_date DATE,
    quantity_change INT
);
```

Here, inventory_id is set to auto-increment for each new record added to inventory_logs.

Task:
Create a new table called inventory_logs in the CodeKraft database. The table should include a unique auto-generated inventory_id for each log entry using the GENERATED ALWAYS AS IDENTITY feature.

Question:
Which SQL statements correctly create the inventory_logs table in the CodeKraft database with inventory_id as a GENERATED ALWAYS AS IDENTITY column? (Select all that apply)

- [x]
```
CREATE TABLE inventory_logs (
    inventory_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    product_id INT,
    change_date DATE,
    quantity_change INT
);
```

- [ ]
```
CREATE TABLE inventory_logs (
    log_id SERIAL PRIMARY KEY,
    product_id INT,
    change_date DATE,
    quantity_change INT
);
```

- [ ]
```
CREATE TABLE inventory_logs (
    inventory_id INT GENERATED PRIMARY KEY,
    product_id INT,
    change_date DATE,
    quantity_change INT
);
```

- [ ]
```
CREATE TABLE inventory_logs (
    inventory_id INT AUTO_INCREMENT PRIMARY KEY,
    product_id INT,
    change_date DATE,
    quantity_change INT
);
```

02:36

2.1.22. SQL Exercise

You are working for an e-commerce company that needs to maintain a record of customer orders. The company wants to track all orders placed by customers, including the unique order identifier, the customer placing the order, the date the order was made, the total amount of the order, and the current status of the order (e.g., pending, completed, or canceled).

Table Schema:

Column Name	Data Type	Constraints
order_id	INTEGER	GENERATED ALWAYS AS IDENTITY
customer_id	INTEGER	-
order_date	DATE	-
total_amount	NUMERIC (10, 2)	-
status	VARCHAR (100)	-

Your task is to write the SQL command to create the customer_orders table in PostgreSQL.

Expected Output:
When you write and run the correct command, the mentioned table will be created and the PostgreSQL displays a message like:

```
CREATE TABLE
```

Explorer

user.sql

```
CREATE TABLE customer_orders(
    order_id INT GENERATED ALWAYS AS IDENTITY,
    customer_id INT,
    order_date DATE,
    total_amount NUMERIC(10,2),
    status VARCHAR(100)
    );
```

Setting up environment...

Home Learn Anywhere

2.1.23. GENERATED BY DEFAULT AS IDENTITY Column

00:55

The GENERATED BY DEFAULT AS IDENTITY column in PostgreSQL is used for auto-generating unique values in a specific column, typically for primary keys. It offers more flexibility compared to GENERATED ALWAYS AS IDENTITY, as it allows for manual overrides of the auto-generated values. This feature is particularly useful when you need to maintain a sequence of unique identifiers but also want the option to insert specific values manually.

Command, Syntax, and Examples

Creating a Table with a GENERATED BY DEFAULT AS IDENTITY Column
Syntax:

```
CREATE TABLE table_name (
    column_name data_type GENERATED BY DEFAULT AS IDENTITY,
    ...
);
```

Example:
Creating the customer_feedback table with an identity column:

```
CREATE TABLE customer_feedback (
    feedback_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    customer_id INT,
    feedback_text TEXT
);
```

In this example, feedback_id is an identity column that typically auto-increments but allows for manually setting specific values if needed.

Task:
Create a new table named special_orders in the CodeKraft database. The table should include a special_order_id column that auto-generates unique identifiers most of the time but also allows for manual entry of IDs.

Question:
Which SQL statements correctly create the special_orders table in the CodeKraft database with special_order_id as a GENERATED BY DEFAULT AS IDENTITY column? (Select all that apply)

- [x]
```
CREATE TABLE special_orders (
    special_order_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    product_id INT,
    order_details TEXT
);
```

- [ ]
```
CREATE TABLE special_orders (
    order_id SERIAL PRIMARY KEY,
    product_id INT,
    order_details TEXT
);
```

- [ ]
```
CREATE TABLE special_orders (
    special_order_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    product_id INT,
    order_details TEXT
);
```

- [ ]
```
CREATE TABLE special_orders (
    special_order_id INT,
    product_id INT,
    order_details TEXT,
    PRIMARY KEY (special_order_id)
);
```

2.1.24. SQL Exercise

01:58

You are tasked with creating a table to track customer orders for an online food delivery system. The system should store the order details such as the order ID, customer ID, order date, total amount, and the current status of each order. The table needs to adhere to the following schema:

Table Schema:

Column Name	Data Type	Constraints
order_id	INTEGER	PRIMARY KEY, GENERATED BY DEFAULT AS IDENTITY
customer_id	INTEGER	-
order_date	DATE	-
total_amount	NUMERIC (10, 2)	-
status	VARCHAR (100)	-

Write an SQL statement to create the **orders** table as per the given schema.

Expected Output:
When you write and run the correct command, the mentioned table will be created and the PostgreSQL displays a message like:

```
CREATE TABLE
```

Explorer

user.sql

```
CREATE TABLE orders(
    order_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY
    customer_id INT,
    order_date DATE,
    total_amount NUMERIC(10,2),
    status VARCHAR(100)
);
```

2.1.25. DEFAULT Values

03:32

In PostgreSQL, creating a table with columns that have default values is a crucial aspect of database design. This feature allows you to set predefined values for a column, which the database automatically uses if no specific value is provided during data insertion. Default values are particularly useful for columns that often contain common values, such as status indicators, timestamps, or configuration flags.

Command, Syntax, and Examples

Creating a Table with Default Value Columns
Syntax:

```
CREATE TABLE table_name ( column1_name column1_datatype DEFAULT default_value, ... );
```

Example:
Creating the customers table with a default membership_status value:

```
CREATE TABLE customers (
  customer_id SERIAL PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100),
  membership_status VARCHAR(20) DEFAULT 'ACTIVE',
  join_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

In this example, new rows in the users table will have a status of 'ACTIVE' and a created_at timestamp of the current date and time if no other values are specified.

Example:

```
CREATE TABLE product_reviews (
  review_id SERIAL PRIMARY KEY,
  product_id INT,
  customer_id INT,
  review_text TEXT,
```

```
CREATE TABLE order_logs (
    log_id SERIAL PRIMARY KEY,
    order_id INT,
    status VARCHAR(50),
    log_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE TABLE order_logs (
    log_id SERIAL PRIMARY KEY,
    order_id INT,
    status VARCHAR(50),
    log_date DATE DEFAULT CURRENT_DATE
);
```

```
CREATE TABLE order_logs (
    log_id SERIAL PRIMARY KEY,
    order_id INT,
    status VARCHAR(50),
    log_date TIMESTAMP DEFAULT NOW()
);
```

```
CREATE TABLE order_logs (
    log_id INT,
    order_id INT,
    status VARCHAR(50) DEFAULT 'PENDING',
    log_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2.1.26. SQL Exercise

01:58

You are working on the development of a new order management system for an e-commerce platform. The system needs to store the details of each order placed by customers, including the unique order ID, customer ID, the date the order was placed, the total amount of the order, and the current status of the order (e.g., "Processing," "Shipped," "Delivered"). The system should automatically assign the current date to the order_date when an order is placed.

You have been provided with the following table schema for creating the orders table in the database:

Table Schema:

Column Name	Data Type	Constraints
order_id	INTEGER	PRIMARY KEY
customer_id	INTEGER	-
order_date	DATE	DEFAULT CURRENT_DATE
total_amount	NUMERIC (10, 2)	-
status	VARCHAR (100)	-

Write the SQL statement to create the **orders** table based on the given schema.

Expected Output:
When you write and run the correct command, the mentioned table will be created and the PostgreSQL displays a message like:

```
CREATE TABLE
```

Explorer

user.sql

```
CREATE TABLE orders(
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE DEFAULT CURRENT_DATE,
    total_amount NUMERIC(10,2),
    status VARCHAR(100)
);
```

2.1.27. SELECT Command - Select all rows and columns

00:11

Selecting all rows and all columns from a table in PostgreSQL is a fundamental database operation, often used to retrieve the entire dataset from a table. This action is performed using the SELECT * FROM statement, where * is a wildcard that represents all columns in the table. It is a common and efficient way to query the full contents of a table for review, analysis, or data processing.

Command, Syntax, and Examples

Selecting All Rows and Columns
Syntax:

```
SELECT * FROM table_name;
```

Example:
To retrieve all data from CodeKraft's customers table:

```
SELECT * FROM customers;
```

This command fetches every column and every row from the customers table.

Task:
In the CodeKraft database, you are asked to retrieve the complete dataset from the products table, including all rows and all columns.

Table Schema for products

Column Name	Data Type	Description
product_id	SERIAL	Primary key for products
name	VARCHAR(255)	Name of the product

- [x] `SELECT * FROM products;`
- [ ] `FETCH ALL FROM products;`
- [ ] `GET ALL FROM products;`
- [ ] `SELECT ALL COLUMNS FROM products;`

Home Learn Anywhere

2.1.28. SQL Exercise

07:57

You are working with an inventory management system for a retail store. The store keeps track of various products in its inventory, including details like the product name, category, quantity, and price. Your manager has requested a report that includes all available products in the inventory. You are tasked with writing an SQL query to fetch all the rows and columns from the **inventory** table to provide a comprehensive overview of the store's current stock.

Inventory Table Design:
Schema

Column Name	Data Type	Constraints
item_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot

item_id	name	category	quantity	price
1	Desk Lamp	Lighting	10	29.99
2	Chair	Furniture	5	89.99
3	Notebook	Stationery	20	3.99

Explorer

user.sql

```
1
2   SELECT * FROM inventory;
```

2.1.29. SELECT Command - Retrieve Specific Columns

00:20

Selecting specific columns from a table in PostgreSQL is a key operation in database querying. It allows you to retrieve only the necessary data, rather than the entire dataset, making the query more efficient and the results easier to handle. This selective approach is particularly useful for large tables or when only certain pieces of information are required for a specific analysis or report.

Command, Syntax, and Examples

Selecting Specific Columns
Syntax:

```
SELECT column1, column2, ... FROM table_name;
```

Example:
In CodeKraft's customers table, to retrieve just the customer's name and email:

```
SELECT name, email FROM customers;
```

This command fetches only the name and email columns from the customers table.

Task:
In the CodeKraft database, you are tasked with retrieving the product_id, name, and price columns from the products table.

Table Schema for products

Column Name	Data Type	Description
product_id	SERIAL	Primary key for products
name	VARCHAR(255)	Name of the product

- [x] `SELECT product_id, name, price FROM products;`
- [ ] `SELECT * FROM products;`
- [ ] `SELECT product_id, category FROM products;`
- [ ] `FETCH product_id, name, price FROM products;`

Home Learn Anywhere

2.1.30. SQL Exercise

02:13

You are managing an online store's product catalog and have been asked to prepare a pricing report for customers. The report should display only the names and prices of all the products currently listed in the database. This information will be used to update the store's website and ensure accurate pricing for customers.

Table Schema for products

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR (255)	NOT NULL
category	VARCHAR (255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot for products

product_id	name	category	quantity	price
1	Desk Lamp	Lighting	10	29.99
2	Office Chair	Furniture	5	89.99
3	Notebook	Stationery	20	3.99

Write an SQL query to select the name and price columns from the products table.

Explorer

user.sql

```
1  SELECT name,price FROM products;
```

2.1.31. AS for Column Aliasing

00:34

In PostgreSQL, selecting specific columns with an alias name involves renaming the columns in the output of a query using the AS keyword. This approach is useful for improving the readability of query results, especially when dealing with complex or less descriptive column names. Aliases are temporary and only affect the output of the query.

Command, Syntax, and Examples

Selecting Specific Columns with Alias Names
Syntax:

```
SELECT column1 AS alias1, column2 AS alias2, ... FROM table_name;
```

Example:
In CodeKraft's customers table, to retrieve the customer's name and email with aliases:

```
SELECT name AS customer_name, email AS customer_email FROM customers;
```

This command fetches the name and email columns from the customers table, but in the output, these columns will be labelled as customer_name and customer_email, respectively.

Task:
In the CodeKraft database, you need to retrieve the product_id and name columns from the products table, but in the query result, these columns should be labelled as ID and ProductName, respectively.

Table Schema for products

Column Name	Data Type	Description
product_id	SERIAL	Primary key for products
name	VARCHAR(255)	Name of the product

- [x] `SELECT product_id AS ID, name AS ProductName FROM products;`
- [ ] `SELECT product_id, name FROM products AS ID, ProductName;`
- [ ] `SELECT ID AS product_id, ProductName AS name FROM products;`

You are managing the product database for an e-commerce platform. The products table stores information about each product, including its ID, name, category, available quantity, and price.

To generate a report for the sales team, you need to write an SQL query to retrieve the following columns:

- The product_id, renamed as ID.
- The name, renamed as ProductName.
- The price, renamed as UnitPrice.

Use column aliases in your query to rename these columns for better clarity in the report.

Table Schema for products:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot for products:

product_id	name	category	quantity	price
1	Desk Lamp	Lighting	10	29.99
2	Office Chair	Furniture	5	89.99
3	Notebook	Stationery	20	3.99

Write an SQL query to select the product_id, name and price columns from the products table as specified.

Expected Output:

user.sql

```
SELECT product_id AS ID, name AS ProductName, price AS UnitPrice FROM products;
```

Using a WHERE clause in a PostgreSQL SELECT statement is fundamental for querying data based on specific conditions. This clause filters the records and returns only those rows that meet the specified criteria, allowing for more precise and targeted data retrieval. It's an essential tool for data analysis and reporting, ensuring that only relevant data is extracted from the database.

Command, Syntax, and Examples

Selecting Rows with a WHERE Clause
Syntax:

```
SELECT column1, column2, ... FROM table_name WHERE condition;
```

Example:
In CodeKraft's orders table, to select orders placed after a certain date:

```
SELECT order_id, customer_id, order_date FROM orders WHERE order_date > '2021-01-01';
```

This command fetches the order ID, customer ID, and order date for all orders placed after January 1, 2021.

Task:
In the CodeKraft database, you need to retrieve all orders from the **orders** table that have a total_amount greater than 500. The orders table includes columns for order_id, customer_id, order_date, status, and total_amount.

Table Schema for orders

Column Name	Data Type	Description
order_id	SERIAL	Primary key for orders
customer_id	INT	ID of the customer placing the order
order_date	DATE	Date when the order was placed
status	VARCHAR(50)	Status of the order
total_amount	NUMERIC(10,2)	Total amount of the order

- [ ] `SELECT * FROM orders WHERE total_amount > 500;`
- [x] `SELECT order_id, customer_id, total_amount FROM orders WHERE total_amount > 500;`
- [ ] `SELECT order_id FROM orders WHERE total_amount < 500;`
- [ ] `SELECT customer_id, order_date, status FROM orders WHERE total_amount = 500;`

2.1.34. SQL Exercise

06:44

You are working for an online retailer managing a database of products. The company has decided to launch a promotional campaign offering discounts on items in the "Electronics" category. To prepare for the campaign, you need to retrieve all information about the products in this category from the **products** table.

Table Schema for products

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot for products

product_id	name	category	quantity	price
1	Wireless Mouse	Electronics	100	19.99
2	HDMI Cable	Electronics	50	9.99
3	Desk Chair	Furniture	30	59.99
4	USB Flash Drive	Electronics	75	14.99
5	Notebook	Stationery	200	2.99

Write an SQL query that selects all columns from the **products** table for items in the **"Electronics"** category.

Explorer

user.sql

SUBMIT

```
1  SELECT product_id, name, category, quantity, price FROM products WHERE
   category = 'Electronics'
```

2.1.35. WHERE with AND

01:07

In PostgreSQL, using a WHERE clause combined with an AND condition in a SELECT statement allows for more precise querying by filtering rows based on multiple criteria. This approach is essential for extracting specific data that meets several conditions simultaneously, making it a powerful tool for detailed data analysis and reporting.

Command, Syntax, and Examples

Selecting Rows with a WHERE Clause and an AND Condition
Syntax:

```
SELECT column1, column2, ... FROM table_name WHERE condition1 AND condition2;
```

Example:
In CodeKraft's orders table, to select orders that were placed after a certain date and have a specific status:

```
SELECT order_id, customer_id, order_date FROM orders WHERE order_date > '2021-01-01' AND status = 'De
```

This command fetches the order ID, customer ID, and order date for all orders placed after January 1, 2021, and have a status of 'Delivered'.

Task:
In the CodeKraft database, you are tasked with retrieving data from the customers table for customers who live in 'New York' and have joined after January 1, 2020. The customers table includes columns for customer_id, name, city, and join_date.

Table Schema for customers

Column Name	Data Type	Description
customer_id	SERIAL	Primary key for customers
name	VARCHAR(255)	Name of the customer
city	VARCHAR(100)	City where the customer lives
join_date	DATE	Date when the customer joined

- [ ] `SELECT * FROM customers WHERE city = 'New York' AND join_date > '2020-01-01';`
- [ ] `SELECT name, city, join_date FROM customers WHERE city = 'New York' AND join_date < '2020-01-01'`
- [x] `SELECT customer_id, name FROM customers WHERE city = 'New York' AND join_date > '2020-01-01';`
- [ ] `SELECT name, join_date FROM customers WHERE city = 'Los Angeles' AND join_date > '2020-01-01'`

2.1.36. SQL Exercise

03:30

You are an inventory manager for an online electronics store. The store is planning a clearance sale to promote budget-friendly products in the "Electronics" category. As part of the preparation, you need to identify all "Electronics" products priced under 100. This information will be used to create a list of sale items for the website.

Table Schema for products:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot for products:

product_id	name	category	quantity	price
1	Wireless Mouse	Electronics	100	19.99
2	HDMI Cable	Electronics	50	9.99
3	Desk Chair	Furniture	30	59.99
4	USB Flash Drive	Electronics	75	14.99
5	Notebook	Stationery	200	2.99
6	Gaming Console	Electronics	20	299.99

te an SQL query to retrieve all columns from the products table where the category is "Electronics" and the price

Explorer

user.sql

SUBMIT

```
1  SELECT * FROM products WHERE category = 'Electronics' and price<100;
```

2.1.37. WHERE with OR

01:09

Selecting rows with a WHERE clause combined with an OR condition in PostgreSQL is a crucial operation for querying data based on alternative criteria. This technique allows for the retrieval of records that meet either one of multiple conditions, enhancing the flexibility of data queries. It's particularly useful for scenarios where you want to include a wider range of data that satisfies any one of several criteria.

Command, Syntax, and Examples

Selecting Rows with a WHERE clause and an OR condition
Syntax:

```
SELECT column1, column2, ... FROM table_name WHERE condition1 OR condition2;
```

Example:
In CodeKraft's products table, to select products that are either in the 'Electronics' category or priced above 100:

```
SELECT product_id, name, price, category FROM products WHERE category = 'Electronics' OR price > 100;
```

This command fetches the product ID, name, price, and category for all products classified as 'Electronics' or having a price greater than 100.

Task:
In the CodeKraft database, you are to retrieve data from the **orders** table for orders that were either placed in 2021 or have a total amount greater than 500. The orders table includes columns for order_id, customer_id, order_date, status, and total_amount.

Table Schema for orders:

Column Name	Data Type	Description
order_id	SERIAL	Primary key for orders
customer_id	INT	ID of the customer placing the order
order_date	DATE	Date when the order was placed
status	VARCHAR(50)	Status of the order

- [ ] `SELECT order_id, customer_id, order_date, status, total_amount FROM orders WHERE order_date >= '`
- [ ] `SELECT order_id, total_amount FROM orders WHERE order_date < '2021-01-01' OR total_amount > 500;`
- [ ] `SELECT order_id, customer_id, order_date FROM orders WHERE order_date >= '2021-01-01' AND order_`
- [x] `SELECT * FROM orders WHERE order_date >= '2021-01-01' AND order_date <= '2021-12-31' OR total_a`

2.1.38. SQL Exercise

01:53

You are a logistics manager for an e-commerce platform. To optimize delivery schedules and customer satisfaction, you need to generate a report on specific orders. The report should include:

- Orders where the total amount exceeds $50, indicating high-value transactions.
- Orders where the status is "Delivered," regardless of the total amount, to monitor fulfilled orders.

This report will be used to analyze successful deliveries and prioritize customer feedback collection.

Table Schema for orders:

Column Name	Data Type	Constraints
order_id	SERIAL	PRIMARY KEY
customer_id	INTEGER	-
order_date	DATE	NOT NULL
total_amount	NUMERIC(10,2)	-
status	VARCHAR(100)	-

ata Snapshot for orders:

order_id	customer_id	order_date	total_amount	status
1	1	2023-12-01	1059.97	Delivered
2	2	2023-12-02	79.96	Shipped
	3	2023-12-03	58.98	Processing
	4	2023-12-04	109.98	Delivered
	5	2023-12-05	19.99	Shipped

Explorer

user.sql

SUBMIT

```
1  SELECT * FROM orders WHERE total_amount >= 50 OR status = 'Delivered';
```

2.1.42. SQL Exercise

02:23

You are responsible for planning a clearance sale for non-electronics items. The criteria for selecting products to include in the sale are:

- The product must not belong to the 'Electronics' category.
- The product must have a price below $100.

Your task is to generate a list of all products meeting these criteria to finalize the clearance sale items.

Table Schema for products:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot for products:

product_id	name	category	quantity	price
1	Wireless Mouse	Electronics	100	19.99
2	HDMI Cable	Electronics	50	9.99
3	Leather Sofa	Furniture	15	599.99
4	USB Flash Drive	Electronics	75	14.99
5	Office Desk	Furniture	25	199.99
6	Gaming Console	Electronics	20	299.99

Explorer

user.sql

SUBMIT

```
1  SELECT * FROM products WHERE category != 'Electronics' AND price<100;
```

2.1.43. LIKE Clause

02:25

In PostgreSQL, the LIKE clause combined with the wildcards _ (representing a single character) and % (representing any sequence of characters) is used for pattern matching in string data. This feature is essential for searching text data based on specific patterns, making it invaluable for filtering results in various text-based queries.

Command, Syntax, and Examples

Using LIKE with _ and % for Pattern Matching
Syntax:

```
SELECT column1, column2, ... FROM table_name WHERE column_name LIKE pattern;
```

Example 1 (Using %):
In CodeKraft's customers table, to select customers whose names start with 'A':

```
SELECT customer_id, name FROM customers WHERE name LIKE 'A%';
```

Example 2 (Using _):
To select customers whose names have any character followed by 'an':

```
SELECT customer_id, name FROM customers WHERE name LIKE '_an%';
```

Example 3 (Using both _ and %): In CodeKraft's products table, to select products whose names have any two characters, followed by 'Pro', and then any sequence of characters:

```
SELECT product_id, name FROM products WHERE name LIKE '__Pro%';
```

sk:
the CodeKraft database, you want to retrieve products from the **products** table whose names start with one aracter, followed by 'an', and then any sequence of characters. The **products** table includes columns for oduct_id, name, price, and category.

le Schema for products

	Data Type	Description

- [x] `SELECT * FROM products WHERE name LIKE '_an%';`
- [ ] `SELECT product_id, name, price FROM products WHERE name LIKE 'an%';`
- [ ] `SELECT name FROM products WHERE name LIKE '_an';`
- [ ] `SELECT category, name FROM products WHERE name LIKE '%an%';`

2.1.44. SQL Exercise

03:16

The furniture department of your company is launching a special promotional campaign to highlight all products that include the word "Desk" in their name. To prepare for this campaign, you need to retrieve a list of all such products.

Table Schema for products:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot for products:

product_id	name	category	quantity	price
1	Desk Lamp	Lighting	10	29.99
2	Office Desk	Furniture	5	89.99
3	Writing Desk	Furniture	7	149.99
4	Study Desk	Furniture	12	79.99
5	Computer Desktop	Electronics	8	499.99
6	Reception Desk	Furniture	3	299.99

...e an SQL query to retrieve all columns from the products table for products whose name contains the word

Explorer

user.sql

SUBMIT

```
1  SELECT * FROM products WHERE name LIKE '%Desk%';
```

2.1.45. SQL Exercise

09:02

A company wants to perform a detailed analysis of their products based on specific patterns in their names. This analysis will help in categorizing and promoting products with unique naming structures.

Write an SQL query to retrieve all columns from the products table for rows where the name column satisfies any of the following conditions:

- Contains any three characters followed by the word 'Desk'.
- Starts with the letter 'C'.
- Ends with the letters 'ir'.
- Contains the substring 'od' anywhere in the middle of the word.

Table Schema for products:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

ata Snapshot for products:

product_id	name	category	quantity	price
1	Desk Lamp	Lighting	10	29.99
2	Chair	Furniture	5	89.99
3	Monitor	Electronics	20	199.99
4	Office Desk	Furniture	12	79.99

Explorer

user.sql

SUBMIT

```
1  SELECT * FROM products WHERE name LIKE 'C%' OR name LIKE '%ir' OR name LIKE
   '%od%' OR name LIKE '___Desk%';
```

2.1.46. IN Clause

01:28

The IN clause in PostgreSQL is used within a SELECT statement to filter data based on a list of specified values. This approach is crucial when you need to retrieve rows that match any one of several values in a specific column. It's a powerful tool for querying datasets where multiple specific criteria are needed, enhancing the efficiency and simplicity of the query.

Command, Syntax, and Examples

Selecting Rows with an IN Clause
Syntax:

```
SELECT column1, column2, ... FROM table_name WHERE column_name IN (value1, value2, ...);
```

Example:
In CodeKraft's products table, to select products that belong to either the 'Electronics' or 'Clothing' categories:

```
SELECT product_id, name, category FROM products WHERE category IN ('Electronics', 'Clothing');
```

This command fetches the product ID, name, and category for products categorized as 'Electronics' or 'Clothing'.

Task:
In the CodeKraft database, you are to retrieve data from the orders table for orders that have a status of either 'Delivered', 'Pending', or 'Cancelled'. The orders table includes columns for order_id, customer_id, order_date, status, and total_amount.

Table Schema for orders:

Column Name	Data Type	Description
order_id	SERIAL	Primary key for orders
customer_id	INT	ID of the customer placing the order
order_date	DATE	Date when the order was placed
status	VARCHAR(50)	Status of the order
total_amount	NUMERIC(10,2)	Total amount of the order

- [x] `SELECT * FROM orders WHERE status IN ('Delivered', 'Pending', 'Cancelled');`
- [x] `SELECT * FROM orders WHERE status = 'Delivered' OR status = 'Pending' OR status = 'Cancelled';`
- [ ] `SELECT customer_id, order_date, total_amount FROM orders WHERE status IN ('Shipped', 'Processing`
- [ ] `SELECT status, total_amount FROM orders WHERE status NOT IN ('Delivered', 'Pending', 'Cancelled`

2.1.47. SQL Exercise

03:49

A supply chain team is conducting an analysis of the store's inventory to prepare for upcoming promotional campaigns. The team needs to focus on items that belong to the categories **'Electronics'**, **'Furniture'**, and **'Stationery'**, as these categories are considered high-priority for their sales strategy.

Write an SQL query to retrieve all details of **products** from the products table that are classified under the categories **'Electronics'**, **'Furniture'**, or **'Stationery'**. Use the IN clause to filter the results based on these categories.

Table Schema for products:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

ata Snapshot for products:

product_id	name	category	quantity	price
1	Desk Lamp	Lighting	10	29.99
2	Chair	Furniture	5	89.99
	Notebook	Stationery	20	3.99
	Laptop	Electronics	8	499.99
	Mouse	Electronics	15	15.99

Explorer

user.sql

SUBMIT

```
1  SELECT * FROM products WHERE category IN
   ('Electronics','Furniture','Stationery');
```

2.1.48. NOT IN Clause

00:18

The NOT IN clause in PostgreSQL is used within a SELECT statement to exclude rows based on a set of specified values. This clause is essential for filtering out specific data, particularly when you need to exclude rows that match any one of several values in a particular column. It's a valuable tool for cases where you want to focus on data that does not meet certain criteria.

Command, Syntax, and Examples

Selecting Rows with a NOT IN Clause
Syntax:

```
SELECT column1, column2, ... FROM table_name WHERE column_name NOT IN (value1, value2, ...);
```

Example:
In CodeKraft's customers table, to select customers who are not located in either 'New York' or 'Los Angeles':

```
SELECT customer_id, name, city FROM customers WHERE city NOT IN ('New York', 'Los Angeles');
```

This command fetches the customer ID, name, and city for all customers whose city is neither 'New York' nor 'Los Angeles'.

Task:
n the CodeKraft database, you need to retrieve data from the products table for all products that are not in the ategories 'Electronics' or 'Clothing'. The products table includes columns for product_id, name, price, and ategory.

able Schema for products:

Column Name	Data Type	Description
product_id	SERIAL	Primary key for products
name	VARCHAR(255)	Name of the product
rice	NUMERIC(10,2)	Price of the product
ategory	VARCHAR(100)	Category of the product

- [x] `SELECT * FROM products WHERE category NOT IN ('Electronics', 'Clothing');`
- [ ] `SELECT product_id, name FROM products WHERE category IN ('Electronics', 'Clothing');`
- [ ] `SELECT * FROM products WHERE NOT category IN ('Electronics', 'Clothing');`
- [x] `SELECT category, price FROM products WHERE category NOT LIKE 'Electronics' AND category NOT LIKE`

2.1.49. SQL Exercise

01:48

The store manager is analyzing products that fall outside the primary promotional categories: 'Electronics', 'Furniture', and 'Stationery'. These excluded items, such as specialty products or niche items, will be considered for separate marketing strategies or clearance sales.

Write an SQL query to retrieve all product details from the products table where the products do not belong to the categories 'Electronics', 'Furniture', or 'Stationery'. Use the NOT IN clause to filter the records for this purpose

Table Schema for products:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot for products:

product_id	name	category	quantity	price
1	Desk Lamp	Lighting	10	29.99
2	Chair	Furniture	5	89.99
3	Notebook	Stationery	20	3.99
4	Laptop	Electronics	8	499.99
5	Mouse	Electronics	15	15.99

Explorer

user.sql

SUBMIT

```
1  SELECT * FROM products WHERE category NOT IN
   ('Electronics','Furniture','Stationery');
```

2.1.50. BETWEEN Clause

00:31

The BETWEEN clause in PostgreSQL is used within a SELECT statement to filter rows within a specific range. This clause is crucial for querying data based on a range of values, whether numeric, date/time, or text. It simplifies queries that require data within a certain scope, enhancing readability and efficiency.

Command, Syntax, and Examples

Selecting Rows with a BETWEEN Clause
Syntax:

```
SELECT column1, column2, ... FROM table_name WHERE column_name BETWEEN value1 AND value2;
```

Example:
In CodeKraft's orders table, to select orders placed between January 1, 2021, and December 31, 2021:

```
SELECT order_id, customer_id, order_date FROM orders WHERE order_date BETWEEN '2021-01-01' AND '2021-1
```

This command fetches the order ID, customer ID, and order date for all orders placed in the year 2021.

Task:
In the CodeKraft database, you need to retrieve data from the products table for products with a price range between $50 and $100. The products table includes columns for product_id, name, price, and category.

Table Schema for products:

Column Name	Data Type	Description
product_id	SERIAL	Primary key for products
name	VARCHAR(255)	Name of the product
price	NUMERIC(10,2)	Price of the product
category	VARCHAR(100)	Category of the product

estion:
ich SQL statements correctly select rows from the products table in the CodeKraft database for products priced

- [x] `SELECT * FROM products WHERE price BETWEEN 50 AND 100;`
- [x] `SELECT * FROM products WHERE price >= 50 AND price <= 100;`
- [ ] `SELECT name, category FROM products WHERE price NOT BETWEEN 50 AND 100;`
- [ ] `SELECT product_id, name, price FROM products WHERE price > 50 AND price < 100;`

2.1.51. SQL Exercise

02:14

The store manager is reviewing products within a mid-range price bracket to feature in an upcoming budget-friendly sale. The focus is on items priced between $10 and $50 (inclusive), as these products are expected to attract customers looking for affordable options.

Write an SQL query to retrieve all product details from the products table where the price falls between $10 and $50, inclusive. Use the BETWEEN clause to filter the records within this price range.

Table Schema for products:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot for products:

product_id	name	category	quantity	price
1	Desk Lamp	Lighting	10	29.99
2	Chair	Furniture	5	89.99
3	Notebook	Stationery	20	3.99
4	Laptop	Electronics	8	499.99
5	Mouse	Electronics	15	15.99
6	Shelf	Furniture	12	79.99

Explorer

user.sql

SUBM

```
1  SELECT * FROM products WHERE price BETWEEN 10 AND 50;
```

2.1.52. ORDER BY Clause

01:02

The ORDER BY clause in PostgreSQL is used within a SELECT statement to sort the result set in either ascending or descending order. By default, when ORDER BY is used without specifying the order, it sorts the results in ascending order. This functionality is fundamental for organizing data in a meaningful sequence, such as sorting products by price, orders by date, or names in alphabetical order.

Command, Syntax, and Examples

Using ORDER BY in Ascending Order
Syntax:

```
SELECT column1, column2, ... FROM table_name ORDER BY column_name [ASC];
```

Example:
In CodeKraft's products table, to sort products by price in ascending order:

```
SELECT product_id, name, price FROM products ORDER BY price ASC;
```

This command fetches the product ID, name, and price of all products and sorts the result set by the price column in ascending order.

Task:
In the CodeKraft database, you are tasked with retrieving data from the customers table and sorting the results by the customer's join date in ascending order. The customers table includes columns for customer_id, name, city, and join_date.

Table Schema for customers:

Column Name	Data Type	Description
customer_id	SERIAL	Primary key for customers
name	VARCHAR(255)	Name of the customer
city	VARCHAR(100)	City where the customer lives
join_date	DATE	Date when the customer joined

- [x] `SELECT * FROM customers ORDER BY join_date ASC;`
- [x] `SELECT * FROM customers ORDER BY join_date;`
- [ ] `SELECT customer_id, city, join_date FROM customers ORDER BY join_date DESC;`
- [ ] `SELECT name, city, join_date FROM customers ORDER BY name ASC;`

2.1.53. SQL Exercise

00:57

The Electronics Store Manager is organizing a price-sensitive promotional catalog for customers. To make it easier for buyers to explore affordable options first, the manager wants to list all products in the Electronics category, sorted by their price in **ascending order.**

Write an SQL query to retrieve all columns from the products table for products in the **"Electronics"** category. Ensure the results are ordered by price in ascending order.

Table Schema for products:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot for products:

product_id	name	category	quantity	price
1	Laptop	Electronics	10	799.99
2	Smartphone	Electronics	20	599.99
3	LED TV	Electronics	5	499.99
4	Wireless Headphones	Electronics	15	149.99
5	Gaming Console	Electronics	8	349.99

Explorer

user.sql

SUBM

```
1  SELECT * FROM products WHERE category = 'Electronics' ORDER BY price;
```

2.1.54. ORDER BY - DESC Clause

00:42

The ORDER BY clause in PostgreSQL is used to sort the results of a SELECT query in a specified order. When used with the DESC keyword, it arranges the results in descending order. This sorting method is particularly useful for organizing data from the highest to the lowest value, such as displaying the latest dates first or listing prices from highest to lowest.

Command, Syntax, and Examples

Using ORDER BY in descending order
Syntax:

```
SELECT column1, column2, ... FROM table_name ORDER BY column_name DESC;
```

Example:
In CodeKraft's orders table, to sort the orders by date in descending order:

```
SELECT order_id, order_date FROM orders ORDER BY order_date DESC;
```

This command sorts the orders by their order_date in descending order, showing the most recent orders first.

Task:
In the CodeKraft database, you need to retrieve the names and join dates of customers from the customers table and sort them by their join date in descending order.

Table Schema for customers:

Column Name	Data Type	Description
customer_id	SERIAL	Primary key for customers
name	VARCHAR(255)	Name of the customer
city	VARCHAR(100)	City where the customer lives

- [ ] `SELECT customer_id, name, join_date FROM customers ORDER BY join_date`
- [x] `SELECT name, join_date FROM customers ORDER BY join_date DESC;`
- [ ] `SELECT name, join_date FROM customers ORDER BY name DESC;`
- [ ] `SELECT MIN(join_date) FROM customers;`

2.1.55. SQL Exercise

02:12

The Procurement Department of a global electronics supply company is evaluating high-end electronics for a premium product showcase. They need a report listing all products in the "Electronics" category, sorted from the most expensive to the least expensive. This will help them decide which products to feature prominently.

Write an SQL query to retrieve all columns from the products table for items belonging to the "Electronics" category. Ensure the query orders the results by price in descending order to highlight the most expensive items first.

Table Schema for products:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot for products:

product_id	name	category	quantity	price
1	Laptop	Electronics	10	799.99
2	Smartphone	Electronics	20	599.99
3	LED TV	Electronics	5	499.99
4	Wireless	Electronics	15	149.99

Explorer

user.sql

```
1  SELECT * FROM products
2  WHERE category = 'Electronics'
3  ORDER BY price DESC;
```

2.1.56. LIMIT Clause

01:19

The LIMIT clause in PostgreSQL is used in a SELECT statement to restrict the number of rows returned by a query. This feature is particularly useful when dealing with large datasets, as it allows for querying a subset of data for quick analysis, testing, or when paginating results. The LIMIT clause helps manage and optimize query performance by reducing the amount of data processed and returned.

Command, Syntax, and Examples

Using LIMIT in a SELECT Statement
Syntax:

```
SELECT column1, column2, ... FROM table_name [ORDER BY column_name] LIMIT number;
```

Example:
In CodeTantra's products table, to select the first 10 products:

```
SELECT product_id, name FROM products LIMIT 10;
```

This command fetches the first 10 rows from the products table, listing the product ID and name.

Task:
In the CodeTantra database, retrieve the top 5 orders with the highest order_id values. Ensure:

1. The rows are sorted in descending order of order_id.
2. The result includes only the following columns: order_id, customer_id, and order_date.

Table Schema for **orders** table:

Column Name	Data Type	Description
order_id	SERIAL	Primary key for orders
customer_id	INT	ID of the customer placing the order
order_date	DATE	Date when the order was placed

- `SELECT order_id, customer_id, order_date FROM orders LIMIT 5;`
- `SELECT * FROM orders ORDER BY order_id LIMIT 5;`
- `SELECT order_id, customer_id, order_date FROM orders ORDER BY order_id DES`
- `SELECT order_id, customer_id, order_date FROM orders WHERE order_id <= 5;`

2.1.57. SQL Exercise

02:20

The Sales Team of an online electronics retailer is planning to feature the three most affordable electronics on their website's homepage for a limited-time discount offer. To prepare the showcase, they need a list of the top 3 cheapest products from the **"Electronics"** category.

Write an SQL query to retrieve all columns from the products table for items in the "Electronics" category. Ensure the results are ordered by price in ascending order and **limited to the first 3 products only.**

Table Schema for products:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot for products:

product_id	name	category	quantity	price
1	Laptop	Electronics	10	799.99
2	Smartphone	Electronics	20	599.99
3	LED TV	Electronics	5	499.99
4	Wireless Headphones	Electronics	15	149.99
5	Gaming Console	Electronics	8	349.99

Explorer

user.sql

```sql
SELECT * FROM products
WHERE category = 'Electronics'
ORDER BY price LIMIT 3;
```

2.1.58. EXISTS clause

12:28

The EXISTS clause in PostgreSQL is used within a SELECT statement to check for the existence of rows in a subquery. This clause is crucial for scenarios where the query's focus is on whether related data exists in another table. It's typically used in conjunction with a subquery and provides a way to perform queries based on the existence of related records.

Command, Syntax, and Examples

Selecting Rows with an EXISTS Clause
Syntax:

```
SELECT column1, column2, ... FROM table_name WHERE EXISTS (subquery);
```

Example:
In CodeKraft's database, to select customers who have placed at least one order:

```
SELECT customer_id, name FROM customers WHERE EXISTS (SELECT 1 FROM orders WHERE customers.customer_i
```

This command fetches the customer ID and name for customers who have at least one associated record in the orders table.

Task:
In the CodeKraft database, you need to retrieve data from the products table for products that have been reviewed at least once. Assume there's a product_reviews table with columns review_id, product_id, customer_id, and review_text.

Table Schema for products:

Column Name	Data Type	Description
product_id	SERIAL	Primary key for products
name	VARCHAR(255)	Name of the product
price	NUMERIC(10,2)	Price of the product

- [ ] `SELECT * FROM products WHERE EXISTS (SELECT 1 FROM product_reviews WHERE products.`
- [ ] `SELECT product_id, name FROM products WHERE product_id IN (SELECT product_id FROM`
- [ ] `SELECT name, price FROM products WHERE NOT EXISTS (SELECT 1 FROM product_reviews`
- [x] `SELECT category FROM products WHERE EXISTS (SELECT 1 FROM product_reviews WHERE`

2.1.59. Understanding ALTER Command

00:11

ALTER:

In PostgreSQL, this command is used for altering the table structure, such as,

- To add a column to existing table
- To rename any existing column
- To change datatype of any column or to modify its size.
- To drop a column from the table.

ALTER Command to add a new Column

Using ALTER command we can add a column to any existing table. Following is the syntax,

```
ALTER TABLE table_name ADD COLUMN (column_name datatype);
```

Here is an example for this:

```
ALTER TABLE student ADD COLUMN (address VARCHAR(200));
```

The above command will add a new column address to the table `student`, which will hold data of type varchar which is nothing but string, of length 200.

ALTER Command: Add multiple new Columns

Using ALTER command we can even add multiple new columns to any existing table. Following is the syntax,

```
ALTER TABLE table_name
ADD COLUMN column1 datatype,
ADD COLUMN column2 datatype,
........;
```

Here is an example for this:

```
ALTER TABLE student
ADD COLUMN father_name VARCHAR(60),
ADD COLUMN mother_name VARCHAR(60),
ADD COLUMN dob DATE;
```

The above command will add three new columns to the `student` table

Add Column with default value:

ALTER command can add a new column to an existing table with a default value too. The default value is used

- CREATE
- DROP
- ALTER
- All given options are correct
- RENAME

2.1.60. ALTER TABLE Command - Renaming Table

00:26

Changing the name of a table in PostgreSQL is a straightforward yet crucial operation in database management. Renaming a table can be necessary for various reasons, such as aligning with updated naming conventions, making the name more descriptive, or correcting naming errors.

This operation should be done cautiously, as it can affect queries and applications that depend on the original table name.

Command, Syntax, and Examples
Changing a Table Name
Syntax:

```
ALTER TABLE current_table_name RENAME TO new_table_name;
```

Example:
Renaming the customer_data table to customer_info:

```
ALTER TABLE customer_data RENAME TO customer_info;
```

This command changes the name of the customer_data table to customer_info in the database.

Task:
In the CodeKraft database, you have a table named order_records that needs to be renamed to customer_orders to better reflect its contents.

Question:
Which SQL statement(s) correctly rename the order_records table to customer_orders in the CodeKraft database? (Select all that apply)

- [x] `ALTER TABLE order_records RENAME TO customer_orders;`
- [ ] `RENAME TABLE order_records TO customer_orders;`
- [ ] `CHANGE TABLE order_records TO customer_orders;`
- [ ] `UPDATE TABLE NAME order_records TO customer_orders;`

2.1.61. SQL Exercise

01:07

The CodeTantra Development Team is in the process of optimizing their database schema to improve clarity and consistency in table naming. As part of this initiative, they have decided to rename the **catalog_items** table to **products**. This table stores information about the products available on their platform, including their ID, name, category, price, description, and stock quantity.

Write an SQL query to rename the table **catalog_items** to **products** in the CodeTantra database. Ensure that all existing data and schema integrity are preserved during the renaming process.

Table Schema: catalog_items

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(100)	-
price	NUMERIC(10,2)	NOT NULL
description	TEXT	-
stock_quantity	INTEGER	-

Data Snapshot in catalog_items:

product_id	name	category	price	description	stock_quantity
1	Smartphone	Electronics	999.99	High-end smartphone	50
2	T-shirt	Clothing	19.99	Cotton t-shirt	200
3	Backpack	Accessories	49.99	Durable	100

Explorer

user.sql

```
1  -- SQL statement to rename the table
2  ALTER TABLE catalog_items RENAME TO products;
```

2.1.62. ALTER TABLE command - Add a column to a table

01:18

Adding a new column to an existing table in PostgreSQL is a common task in database management. This operation is essential when new data needs to be incorporated into an existing dataset or when modifying the table structure to accommodate changes in data requirements. It allows for flexibility and scalability in database design without disrupting existing data.

Command, Syntax, and Examples
Adding a Column to an Existing Table
Syntax:

```
ALTER TABLE table_name ADD COLUMN new_column_name data_type;
```

Example:
Adding a category column to the existing products table:

```
ALTER TABLE products ADD COLUMN category VARCHAR(100);
```

This command adds a new column named category of type VARCHAR(100) to the products table.

Task:
In the CodeKraft database, you have an existing table named customers. You need to add a new column date_of_birth to this table to store the customers' birth dates.

Question:
Which SQL statement(s) correctly add a date_of_birth column to the existing customers table in the CodeKraft database? (Select all that apply)

- [x] `ALTER TABLE customers ADD COLUMN date_of_birth DATE;`
- [ ] `ADD COLUMN date_of_birth DATE TO customers;`
- [ ] `UPDATE TABLE customers ADD date_of_birth DATE;`
- [ ] `ALTER TABLE customers INSERT COLUMN date_of_birth DATE;`

2.1.63. SQL Exercise

02:51

The customer management system at your organization requires an update to store additional information about customers. To achieve this, the **customers** table in the database needs to include a new column to store the age of each customer. This new column will help the organization better segment and analyze customer demographics.

Write an SQL query to add a new column named **age** of data type INTEGER to the existing customers table. Ensure that the schema modification does not alter or delete any existing data in the table.

Table Schema:

Column Name	Data Type	Constraints
customer_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
email	VARCHAR(255)	NOT NULL
address	TEXT	-
phone_number	VARCHAR(50)	-

Data Snaptshot for customers table:

customer_id	name	email	address	phone_number
1	John Doe	johndoe@email.com	123 Apple St.	123-456-7890
2	Jane Smith	janesmith@email.com	456 Orange Ave.	234-567-8901
3	Bob Johnson	bobj@email.com	789 Banana Rd.	345-678-9012
4	Alice Brown	alice@email.com	321 Cherry Blvd.	456-789-0123

Explorer

user.sql

```
1  --Type SQL here
2  ALTER TABLE customers ADD COLUMN age INT;
```

2.1.64. ALTER TABLE Command - Delete a column from table

00:43

Deleting a column from a table in PostgreSQL is a crucial database management operation. This task is often needed when a specific piece of data becomes irrelevant, or the database schema is being streamlined. While it's a powerful tool for database optimization, it should be used cautiously, as it results in the permanent removal of the column and its data.

Command, Syntax, and Examples

Deleting a column from a table
Syntax:

```
ALTER TABLE table_name DROP COLUMN column_name;
```

Example:
Removing the category column from the products table:

```
ALTER TABLE products DROP COLUMN category;
```

This command deletes the category column and all its data from the products table.

Task:
In the CodeKraft database, there's a need to remove the phone_number column from the existing customers table, as it's no longer required for the business process.

Question:
Which SQL statement(s) correctly delete the phone_number column from the existing customers table in the CodeKraft database? (Select all that apply)

- [x] `ALTER TABLE customers DROP COLUMN phone_number;`
- [ ] `DELETE COLUMN phone_number FROM customers;`
- [ ] `REMOVE COLUMN phone_number FROM TABLE customers;`
- [ ] `ALTER TABLE customers DELETE COLUMN phone_number;`

2.1.65. SQL Exercise

01:26

The inventory management system in your organization is undergoing optimization to simplify its database structure. As part of this update, the **description** column in the **products** table is no longer required for business operations. To streamline the database, this column must be removed.

Write an SQL query to remove the **description** column from the **products** table in the inventory database. Ensure that this operation does not affect the integrity or data in the remaining columns.

Original Table Schema: products

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(100)	-
price	NUMERIC(10,2)	NOT NULL
description	TEXT	-
stock_quantity	INTEGER	-

Data in products Table:

product_id	name	category	price	description	stock_quantity
1	Smartphone	Electronics	999.99	High-end smartphone	50
2	T-shirt	Clothing	19.99	Cotton t-shirt	200
3	Backpack	Accessories	49.99	Durable backpack	100

Explorer

user.sql

```
1  ALTER TABLE products DROP COLUMN description;
```

2.1.66. ALTER TABLE Command - Change the data type of a column

00:56

Changing the data type of a column in PostgreSQL is an essential aspect of database management, particularly when the nature of the data in that column evolves or was initially misconfigured. This operation allows for the modification of the column's data type to better reflect the kind of data stored, ensuring data integrity and optimal database performance.

Command, Syntax, and Examples

Changing the Data Type of a Column
Syntax:

```
ALTER TABLE table_name ALTER COLUMN column_name TYPE new_data_type;
```

Example:
Changing the data type of the price column in the products table from NUMERIC to FLOAT:

```
ALTER TABLE products ALTER COLUMN price TYPE FLOAT;
```

This command alters the price column in the products table to have a data type of FLOAT instead of its previous NUMERIC type.

Task:
In the CodeKraft database, the customer_reviews table has a column rating initially set as VARCHAR. There's a need to change this column's data type to INT as it should store numerical values.

Question:
Which SQL statement(s) correctly change the data type of the rating column to INT in the customer_reviews table in the CodeKraft database? (Select all that apply)

- [ ] `ALTER TABLE customer_reviews MODIFY COLUMN rating INT;`
- [x] `ALTER TABLE customer_reviews ALTER COLUMN rating TYPE INT;`
- [ ] `UPDATE TABLE customer_reviews CHANGE COLUMN rating rating INT;`
- [ ] `ALTER TABLE customer_reviews CHANGE rating rating INT;`

2.1.67. SQL Exercise

02:15

The E-Shop Logistics team is revamping its database structure to better align with financial regulations and improve operational clarity. Currently, the database includes a table named **orders**, which holds customer order information. However, this table needs a more descriptive name. Additionally, the column **total_amount**, which stores the monetary value of orders, is defined with a data type that is insufficient for handling larger amounts accurately.

You are required to perform the following queries:

- **Query 1:** Rename the **orders** table to **client_purchases** to make the table's purpose more explicit.
- **Query 2:** Update the **total_amount** column's data type from **NUMERIC(10, 2)** to **DECIMAL(12, 2)** to comply with financial standards for high-value transactions.

Table Schema:

Column Name	Data Type	Constraints
order_id	SERIAL	PRIMARY KEY
customer_id	INTEGER	-
order_date	DATE	NOT NULL
total_amount	NUMERIC(10,2)	-
status	VARCHAR(100)	-

Sample Data:

order_id	customer_id	order_date	total_amount	status
1	1	2023-12-01	1059.97	Delivered
2	2	2023-12-02	79.96	Shipped
3	3	2023-12-03	58.98	Processing

Explorer

user.sql

```sql
--Query 1: Rename the orders table to client_purchases
ALTER TABLE orders RENAME TO client_purchases;

--Query 2: Update the total_amount column's data type from NUMERIC
DECIMAL(12,2)
ALTER TABLE client_purchases ALTER COLUMN total_amount TYPE DECIMAL
```

2.1.68. ALTER TABLE Command - Renaming a Columns

00:39

Changing a column name in PostgreSQL is a key operation in database management, especially when refining or correcting your database schema. Renaming a column can be necessary for various reasons, such as making the column name more descriptive, aligning with new naming conventions, or correcting a typo. This action requires caution as it might impact existing queries and applications relying on the original column name.

Command, Syntax, and Examples

Changing a Column Name
Syntax:

```
ALTER TABLE table_name RENAME COLUMN current_column_name TO new_column_name;
```

Example:
Renaming the cust_email column to customer_email in the customers table:

```
ALTER TABLE customers RENAME COLUMN cust_email TO customer_email;
```

This command changes the name of the cust_email column in the customers table to customer_email.

Task:
In the CodeKraft database, the orders table has a column named order_date_time. It needs to be renamed to order_date for clarity and consistency.

Question:
Which SQL statement(s) correctly change the name of the order_date_time column to order_date in the orders table in the CodeKraft database? (Select all that apply)

- [ ] `UPDATE TABLE orders CHANGE COLUMN order_date_time TO order_date;`
- [x] `ALTER TABLE orders RENAME COLUMN order_date_time TO order_date;`
- [ ] `ALTER TABLE orders CHANGE order_date_time TO order_date;`
- [ ] `RENAME COLUMN order_date_time TO order_date IN TABLE orders;`

2.1.69. SQL Exercise

03:37

The CodeClear Product Review Team has decided to streamline the **product_reviews** table in their database for better performance and consistency. To achieve this, several schema updates need to be implemented.

You are required to perform the following modifications to the **product_reviews** table in the CodeClear database:

- **Query 1: Rename the table** from **product_reviews** to **reviews** to simplify the table name.
- **Query 2: Modify the data type** of the **rating** column from **INTEGER** to **SMALLINT** for more efficient storage, as ratings are always small values.
- **Query 3: Remove the comment** column as it is deemed unnecessary for the current application.
- **Query 4: Rename the product_id** column to **item_id** to align with the naming convention used across other tables.

Table Schema:

Column Name	Data Type	Constraints
review_id	SERIAL	PRIMARY KEY
product_id	INTEGER	-
customer_id	INTEGER	-
rating	INTEGER	-
comment	TEXT	-

Data for product_reviews:

review_id	product_id	customer_id	rating	comment
1	1	1	5	Excellent smartphone!
2	2	2	4	Comfortable t-shirt.
3	3	3	3	Decent backpack.

Explorer

user.sql

```sql
-- Rename the table from product_reviews to reviews
ALTER TABLE product_reviews RENAME TO reviews;

-- Modify the data type of the rating column from INTEGER to SMALLI
ALTER TABLE reviews ALTER COLUMN rating TYPE SMALLINT;

-- Remove the comment column
ALTER TABLE reviews DROP COLUMN comment;

-- Rename the product_id column to item_id
ALTER TABLE reviews RENAME COLUMN product_id TO item_id;


```

2.1.69. SQL Exercise

03:37

The CodeClear Product Review Team has decided to streamline the **product_reviews** table in their database for better performance and consistency. To achieve this, several schema updates need to be implemented.

You are required to perform the following modifications to the **product_reviews** table in the CodeClear database:

- **Query 1: Rename the table** from **product_reviews** to **reviews** to simplify the table name.
- **Query 2: Modify the data type** of the **rating** column from **INTEGER** to **SMALLINT** for more efficient storage, as ratings are always small values.
- **Query 3: Remove the comment** column as it is deemed unnecessary for the current application.
- **Query 4: Rename the product_id** column to **item_id** to align with the naming convention used across other tables.

Table Schema:

Column Name	Data Type	Constraints
review_id	SERIAL	PRIMARY KEY
product_id	INTEGER	-
customer_id	INTEGER	-
rating	INTEGER	-
comment	TEXT	-

Data for product_reviews:

review_id	product_id	customer_id	rating	comment
1	1	1	5	Excellent smartphone!
2	2	2	4	Comfortable t-shirt.
3	3	3	3	Decent backpack.

Explorer

user.sql

```sql
-- Rename the table from product_reviews to reviews
ALTER TABLE product_reviews RENAME TO reviews;

-- Modify the data type of the rating column from INTEGER to SMALLI
ALTER TABLE reviews ALTER COLUMN rating TYPE SMALLINT;

-- Remove the comment column
ALTER TABLE reviews DROP COLUMN comment;

-- Rename the product_id column to item_id
ALTER TABLE reviews RENAME COLUMN product_id TO item_id;


```

2.1.70. DROP TABLE command

00:05

Deleting a table in PostgreSQL is a significant operation that involves permanently removing a table and all of its data from the database. This action is typically performed when a table becomes obsolete or irrelevant. It's a critical task that should be approached with caution, as once a table is deleted, all the information stored in it is lost and cannot be recovered unless backups are available.

Command, Syntax, and Examples

Deleting a Table
Syntax:

```
DROP TABLE table_name;
```

Example:
Deleting the temporary_data table:

```
DROP TABLE temporary_data;
```

This command removes the temporary_data table from the database, along with all its data.

Task:
In the CodeKraft database, there is a table named old_customer_records that is no longer in use and needs to be deleted.

Question:
Which SQL statement(s) correctly delete the old_customer_records table in the CodeKraft database? (Select all that apply)

- [ ] `DELETE TABLE old_customer_records;`
- [ ] `REMOVE TABLE old_customer_records;`
- [x] `DROP TABLE old_customer_records;`
- [ ] `ALTER TABLE old_customer_records DROP;`

Home Learn Anywhere

12411693.st@lpu.in Support

2.1.71. SQL Exercise

00:47

You are managing a database for a retail company, and due to changes in the business model, the company no longer needs to store customer information in the database. You have been assigned the task of removing the **customers** table from the database.

Write the SQL statement to drop the **customers** table from your PostgreSQL database.

Expected Output:
If you write and execute the correct command, the table will be deleted and the PostgreSQL typically displays a message like:

```
DROP TABLE
```

Explorer

user.sql

```
1  DROP TABLE customers;
```

2.1.72. TRUNCATE TABLE Command

00:50

Deleting all rows from a table while retaining the table structure in PostgreSQL is a common operation, often used in data refresh scenarios, testing, or clearing out data without removing the table itself. This action is achieved using the TRUNCATE command, which quickly removes all records from a table but leaves the table structure and its definitions, such as columns and constraints, intact.

Command, Syntax, and Examples

Deleting All Rows While Keeping Table Structure

Using TRUNCATE
Syntax:

```
TRUNCATE TABLE table_name;
```

Example:
Removing all data from the customer_logs table:

```
TRUNCATE TABLE customer_logs;
```

This command deletes all rows from the customer_logs table but keeps the table itself and its schema.

Using DELETE
Syntax:

```
DELETE FROM table_name;
```

Example:

```
DELETE TABLE customer_logs;
```

Task:
In the CodeKraft database, you need to delete all entries in the session_data table to clear out old user session records, but the table structure must remain for future data.

Question:
Which SQL statements correctly delete all rows from the session_data table while retaining its structure in the

- [x] `DELETE FROM session_data;`
- [ ] `DROP TABLE session_data;`
- [x] `TRUNCATE TABLE session_data;`
- [ ] `REMOVE ALL FROM session_data;`

2.1.73. SQL Exercise

02:02

You are working as a database administrator for a global retail company. The company is currently running a promotional campaign, and you need to refresh the customer data in the system. However, you must ensure that the structure of the customers table remains intact, and only the data is cleared. This action is part of the data preparation process before importing updated customer data.

Your task is to empty all the current records from the customers table while keeping the table structure intact, so that it can later be populated with fresh data.

Table Schema for customers:

Column Name	Data Type	Constraints
customer_id	SERIAL	PRIMARY KEY
name	VARCHAR (255)	NOT NULL
age	INTEGER	-
nationality	VARCHAR (100)	-
address	TEXT	-

Expected Output:
After performing the operation, the customers table will become empty, but its structure (columns and data types) will remain unchanged. No customer data should be present in the table after the operation. And the PostgreSQL typically displays a message like:

```
TRUNCATE TABLE
```

Explorer

user.sql

```
1  TRUNCATE TABLE customers
```

2.1.74. TRUNCATE ... RESTART IDENTITY Command

02:34

In PostgreSQL, when managing tables such as customer records or order details, it's sometimes necessary to delete all data while resetting auto-increment (identity) columns. This is particularly useful in scenarios like refreshing a test database or reinitializing data. The TRUNCATE ... RESTART IDENTITY command accomplishes this by removing all rows and resetting auto-increment counters.

Command, Syntax, and Examples

Deleting All Rows and Resetting Identity Columns
Syntax:

```
TRUNCATE TABLE table_name RESTART IDENTITY;
```

Example:
Clearing all data and resetting auto-increment in the order_details table:

```
TRUNCATE TABLE order_details RESTART IDENTITY;
```

This command will remove all rows from the order_details table and reset the auto-increment counter for the identity column.

Task:
In CodeKraft's database, there's a requirement to clear all data from the product_reviews table and reset the sequence of its primary key (assuming it's an auto-increment field) for a new set of product review data.

Question:
Which SQL statements correctly delete all rows from the product_reviews table and reset the auto-increment sequence of the identity column in the CodeKraft database? (Multiple answers possible)

- [x]
```
TRUNCATE TABLE product_reviews RESTART IDENTITY;
```

- [x]
```
DELETE FROM product_reviews;
ALTER SEQUENCE product_reviews_review_id_seq RESTART WITH 1;
```

- [ ]
```
TRUNCATE TABLE product_reviews;
```

- [ ]
```
DROP TABLE product_reviews;
CREATE TABLE product_reviews (...);
```

2.1.75. SQL Exercise

03:23

You are a database administrator for a large retail company that is preparing for a major inventory update. The **products** table contains information about all current products, including their IDs, names, categories, prices, descriptions, and stock quantities. Before importing the updated product data into the system, you need to clear the existing records from the **products** table while ensuring that the auto-increment value for the **product_id** column resets to its initial value. This will allow the new products to be added smoothly, starting from the first product ID.

Your goal is to execute the necessary operation to clear all the records from the **products** table and reset the auto-increment value for **product_id**.

Table Schema:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(100)	-
price	NUMERIC(10,2)	NOT NULL
description	TEXT	-
stock_quantity	INTEGER	-

Data for products table:

product_id	name	category	price	description	stock_quantity
1	Smartphone	Electronics	999.99	High-end smartphone	50
2	T-shirt	Clothing	19.99	Cotton t-shirt	200
				Durable	

Explorer

user.sql

```
1  DELETE FROM products;
2  ALTER SEQUENCE products_product_id_seq RESTART WITH 1;
```

2.1.76. Including comments in PostgreSQL

01:02

In PostgreSQL, comments are used to provide explanations or documentation within the code itself. PostgreSQL supports two types of comments: single-line comments and multi-line comments.

Single-Line Comments:
Single-line comments start with -- and extend until the end of the line. These comments are used for providing explanations or annotations on a single line of code.
Example:

```
-- This is a single-line comment
SELECT * FROM employees; -- This comment follows a statement
```

Multi-Line Comments:
Multi-line comments are enclosed within /* and */. They can span multiple lines and are useful for providing detailed explanations or for commenting out blocks of code temporarily.
Example:

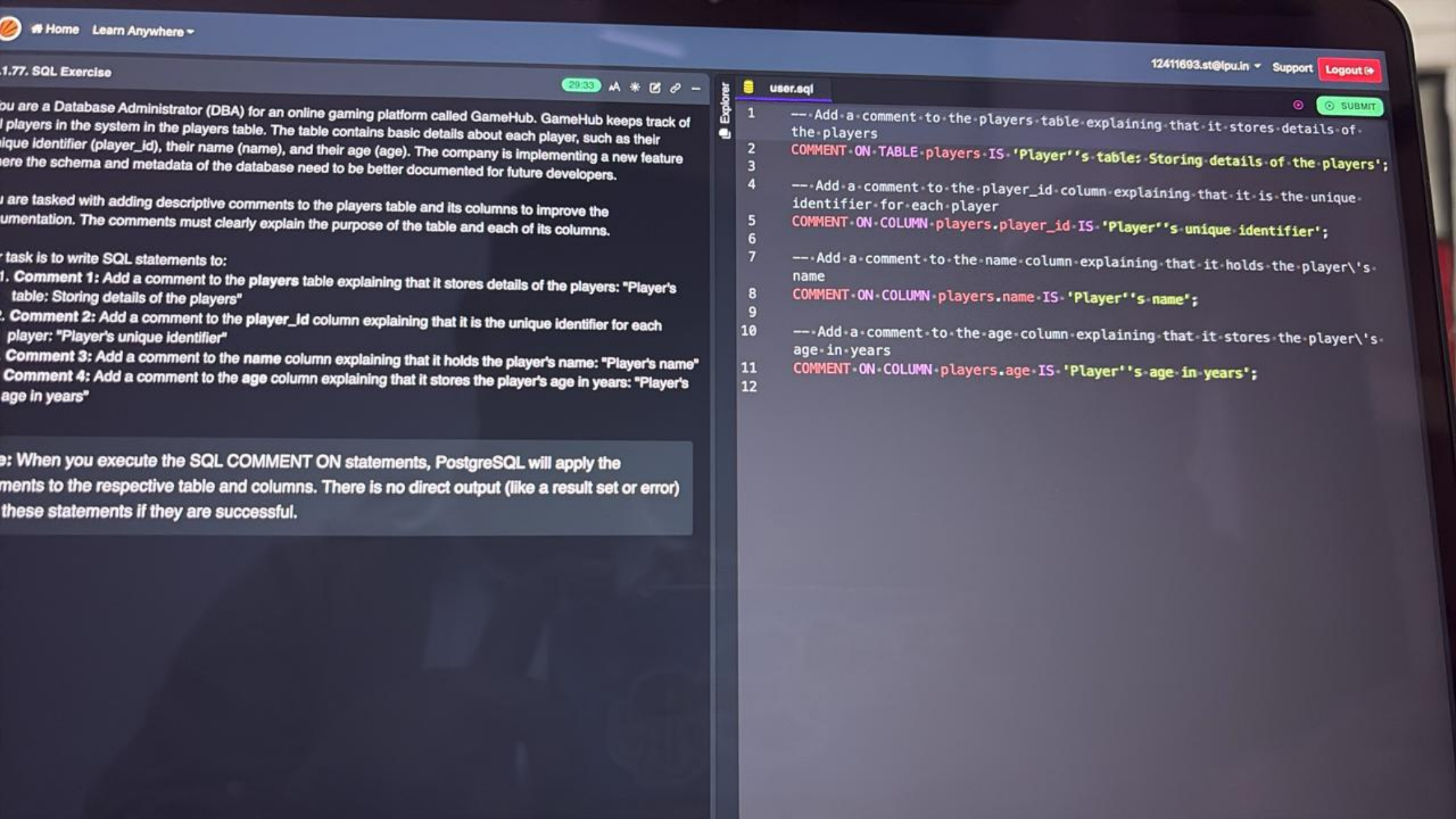
```
/* This is a
multi-line comment */
SELECT * FROM departments;
```

Note: In PostgreSQL, you cannot nest multi-line comments. This means you cannot have one multi-line comment inside another.

Using Comments Effectively:
Comments should be used effectively to document your SQL code, especially for complex queries or stored procedures. Here are some tips on using comments effectively:

- **Explain Complex Logic:** Use comments to explain complex logic or calculations in your queries, making it easier for others to understand your code.
- **Document Changes:** If you make changes to your code, update the comments accordingly to reflect the current logic.
- **Temporary Commenting:** When debugging or testing, you can comment out parts of your code temporarily to isolate issues without deleting the code.
- **Version Control:** In collaborative projects, comments can help team members understand the purpose of specific code sections and changes made over time.
- **Use Descriptive Comments:** Write descriptive comments that provide context and explain the "why" behind

- [ ] Single-line comments in PostgreSQL start with /*.
- [ ] Multi-line comments in PostgreSQL start with --.
- [x] Multi-line comments in PostgreSQL are enclosed within /* and */.
- [x] Nesting of multi-line comments is not supported in PostgreSQL.


Home
Learn Anywhere
1.77. SQL Exercise
29:33
12411693.st@lpu.in
Support
Logout
bu are a Database Administrator (DBA) for an online gaming platform called GameHub. GameHub keeps track of l players in the system in the players table. The table contains basic details about each player, such as their ique identifier (player_id), their name (name), and their age (age). The company is implementing a new feature ere the schema and metadata of the database need to be better documented for future developers.
u are tasked with adding descriptive comments to the players table and its columns to improve the umentation. The comments must clearly explain the purpose of the table and each of its columns.
task is to write SQL statements to:
1. Comment 1: Add a comment to the players table explaining that it stores details of the players: "Player's table: Storing details of the players"
2. Comment 2: Add a comment to the player_id column explaining that it is the unique identifier for each player: "Player's unique identifier"
3. Comment 3: Add a comment to the name column explaining that it holds the player's name: "Player's name"
4. Comment 4: Add a comment to the age column explaining that it stores the player's age in years: "Player's age in years"
e: When you execute the SQL COMMENT ON statements, PostgreSQL will apply the ments to the respective table and columns. There is no direct output (like a result set or error) these statements if they are successful.
Explorer
user.sql
SUBMIT
1 --·Add·a·comment·to·the·players·table·explaining·that·it·stores·details·of·the·players
2 COMMENT·ON·TABLE·players·IS·'Player''s·table:·Storing·details·of·the·players';
3
4 --·Add·a·comment·to·the·player_id·column·explaining·that·it·is·the·unique·identifier·for·each·player
5 COMMENT·ON·COLUMN·players.player_id·IS·'Player''s·unique·identifier';
6
7 --·Add·a·comment·to·the·name·column·explaining·that·it·holds·the·player\'s·name
8 COMMENT·ON·COLUMN·players.name·IS·'Player''s·name';
9
10 --·Add·a·comment·to·the·age·column·explaining·that·it·stores·the·player\'s·age·in·years
11 COMMENT·ON·COLUMN·players.age·IS·'Player''s·age·in·years';
12

You are working on a flight reservation system. The system needs to store essential flight and passenger details for flight management and bookings. You are tasked with creating a table to record flight details and passenger information. The table should track information like flight number, passenger name, origin, destination, and the class of service.

You will also need to alter the table by adding an additional column for seat_number, and then truncate all data when needed.

flight_reservations Table Schema:

Column Name	Data Type	Constraints	Description
flight_no	INT	PRIMARY KEY	A unique identifier for each flight.
passenger_name	VARCHAR(50)	NOT NULL	The name of the passenger.
origin	VARCHAR(50)	-	The origin city of the flight.
destination	VARCHAR(50)	-	The destination city of the flight.
class	VARCHAR(50)	-	The class of service (e.g., Economy, Business, First Class).

Flight Reservation System Requirements:

- **Step 1:** Create a table for storing flight reservation details as mentioned in the above schema
- **Step 2:** Alter the table to include a new column `seat_no` with VARCHAR(10) Data Type
- **Step 3:** Truncate all data from the table

Output:

- After creating the table and altering it, the table will have a new column `seat_num` (VARCHAR(10)) along with the above mentioned columns.
- After truncating the table, all data will be removed, leaving the table empty but still intact. The table schema remains the same, so it can be reused.

Table flight_reservations need to be created as per below schema

user.sql

SUBMIT

```sql
-- CREATE TABLE for storing flight reservation details
CREATE TABLE flight_reservations(
    flight_no INT PRIMARY KEY,
    passenger_name VARCHAR(50) NOT NULL,
    origin VARCHAR(50),
    destination VARCHAR(50),
    class VARCHAR(50)
);

....
-- Alter the table to include a new column
ALTER TABLE flight_reservations ADD COLUMN seat_no VARCHAR(10);

-- Insert values to table
INSERT INTO flight_reservations (flight_no, passenger_name, origin, destination, class, seat_no)
VALUES
    (104, 'Michael Brown', 'Boston', 'Denver', 'First', '1A'),
    (105, 'Emma Wilson', 'Atlanta', 'Dallas', 'Business', '2B'),
    (106, 'Olivia Davis', 'Seattle', 'Chicago', 'Economy', '3C');

-- Truncate all data from the table
TRUNCATE TABLE flight_reservations;
```

Terminal Test cases

2.2.2. DDL - ALTER - Library System Management

02:27

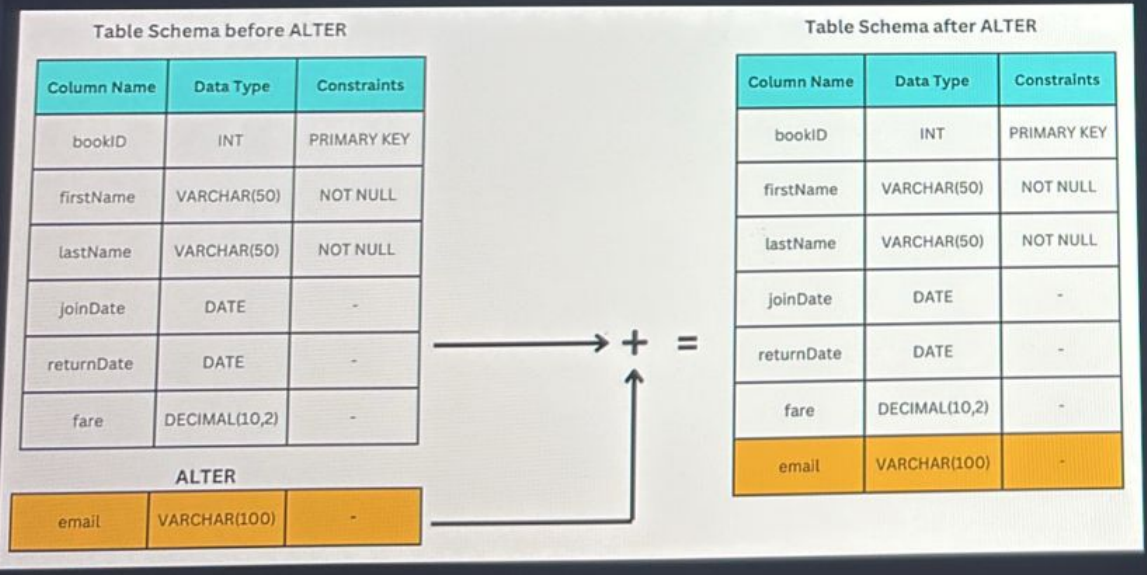
You are enhancing a Library Management System and need to add functionality to contact library members via email. To achieve this, you must update the **library_members** table by adding a new column named **email**. This column will store the email addresses of members and will help the library staff communicate with members about book due dates, events, and notifications.

Requirements:

- The column name should be `email`.
- The column data type should be `VARCHAR(100)`.
- Ensure the column can store up to 100 characters.
- The column should allow NULL values, as not all members may have an email address initially.

Write an SQL statement to implement this requirement.

Note: You are not required to create the table, The table is already available in the database.



Output: The **email** column is added to the `library_members` table.

Explorer

index.sql

```sql
-- Add the email column
ALTER TABLE library_members ADD COLUMN email VARCHAR(100);
```

2.2.3. DDL - Police Department Information Management System

00:45

You are working for the Police Department's Data Management System. Due to a sudden network breach, unauthorized access was gained to the police department's internal records, and sensitive officer information was compromised. As a result, all the data in the `police_officers` table must be removed immediately to prevent further misuse. However, the table structure must remain intact so that new and verified data can be inserted after the security breach has been resolved.

Task:
Your task is to remove all the records from the `police_officers` table as quickly as possible, but keep the table structure intact for future secure data entry.

Table Schema for police_officers:

Column Name	Data Type	Constraints
officerID	INT	PRIMARY KEY
firstName	VARCHAR(50)	NOT NULL
lastName	VARCHAR(50)	NOT NULL
rank	VARCHAR(50)	-
salary	DECIMAL(10,2)	-
dateOfJoining	DATE	-

Data Snapshot:

officerID	firstName	lastName	rank	salary	dateOfJoining
1	John	Doe	Sergeant	50000.00	2010-05-01
2	Jane	Smith	Lieutenant	60000.00	2015-08-10
3	Mark	Johnson	Captain	75000.00	2012-11-23

Explorer

index.sql

```sql
-- Remove all records from the table
TRUNCATE TABLE police_officers;

```

2.2.4. DDL - Weather Monitoring System

01:33

You are working on a Weather Monitoring System that collects real-time weather data from various regions. Recently, the system underwent a major overhaul, and all data from the old `weather_data` table has been successfully migrated to a new structure. As part of the cleanup process, the outdated `weather_data` table must be completely removed from the database to avoid any conflicts with the new design.

Requirements:

- The table name is `weather_data`.
- Ensure the table is dropped from the database without errors.

Write an SQL statement to drop the `weather_data` table from the database.

Expected Output:
The `weather_data` table will be permanently removed from the database.

Note: You are not required to create the table, The table is already available in the database.

Explorer

index.sql

```
1  DROP TABLE weather_data;
```

2.2.5. DDL - Employee Management System

11:06

You are working for a rapidly growing company called CodeTantra, which is expanding its operations. The HR department requires a robust employee management system to maintain detailed records of all employees. This system must store essential employee information, including their personal details, salaries, and hire dates. To ensure accurate record-keeping, you need to create a table named `employees` that adheres to the schema provided below.

`employees` table schema:

Column Name	Data Type	Constraints
employeeID	INT	PRIMARY KEY
firstName	VARCHAR(50)	NOT NULL
lastName	VARCHAR(50)	NOT NULL
birthDate	DATE	-
hireDate	DATE	-
salary	DECIMAL(10, 2)	CHECK (salary >= 0)

Requirements:

- The table name must be **employees.**
- The **employeeID** column will serve as the primary key to uniquely identify each employee.
- The **firstName** and **lastName** columns cannot be null, ensuring that every employee's name is recorded.
- The **salary** column must include a check constraint to ensure only non-negative salary values are allowed.
- Other columns like **birthDate** and **hireDate** are optional and can accept **NULL** values.

Write a SQL query to create the **employees** table with the schema and constraints described above.

Expected Outcome:

The employees table will be successfully created in the database.

- As no data is inserted or selected, running the query will not return any output.

Explorer

index.sql

```sql
CREATE TABLE employees(
    employeeID INT PRIMARY KEY,
    firstName VARCHAR(50) NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    birthDate DATE,
    hireDate DATE,
    salary DECIMAL(10,2) CHECK (salary>=0)
    );

```

2.2.6. DDL - Adding New Column

01:14

You are working on an employee management system for a large corporation. Recently, the company introduced a new department called "Innovation". To track which employees belong to this new department or any other department, you need to modify the existing `employees` table by adding a new column named `department`.

Requirements:

- The **employees** table already exists in the database.
- Add a new column named **department** to the table.
- The column **department** should have the data type **VARCHAR(100)**.
- This column will store the department name for each employee.
- The column can accept **NULL** values initially, as not all employees may be assigned to a department at the time of addition.

employees table schema:

Column Name	Data Type	Constraints
employeeID	INT	PRIMARY KEY
firstName	VARCHAR(50)	NOT NULL
lastName	VARCHAR(50)	NOT NULL
birthDate	DATE	-
hireDate	DATE	-
salary	DECIMAL(10,2)	CHECK (salary >= 0)

Write a SQL query to add the **department** column to the **employees** table.

Expected Output:

- The department column will be added to the employees table successfully.
- The query will execute without returning any result, as it only modifies the table structure.

Explorer

index.sql

```sql
-- Add the new column to the table
ALTER TABLE employees ADD COLUMN department VARCHAR(100);

```

2.2.7. Modifying Table Structures and Data: Alterations in SQL - 2

01:56

You are working with a table named emp in the database, which stores employee information. Perform the following tasks to update the table structure as required:

emp Table Schema (Before Modification):

Column Name	Data Type	Constraints
employeeID	INT	PRIMARY KEY
employeeName	VARCHAR(100)	-
salary	DECIMAL(10, 2)	-
depttname	VARCHAR(100)	-

Tasks 1: Add a New Column

Add a new column named **designation** to the **emp** table.

- **Column Name:** designation
- **Data Type:** VARCHAR(100)

Expected Outcome: The new column designation will be successfully added to the emp table.

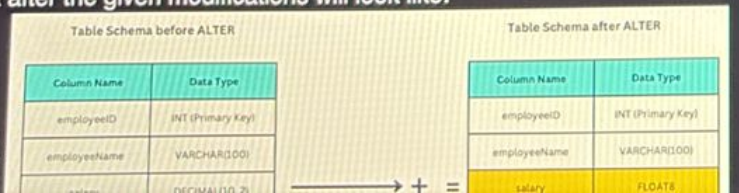
Task 2: Change the Data Type of salary

Change the data type of the salary column from DECIMAL(10, 2) to FLOAT8.

- **Column Name:** salary
- **New Data Type:** FLOAT8

Expected Outcome: The data type of the salary column will be updated to FLOAT8, allowing it to store floating-point values.

The emp table schema after the given modifications will look like:



Explorer

index.sql

```
1  --·Add·the·designation·Column
2  ALTER·TABLE·emp·ADD·COLUMN·designation·VARCHAR(100);
3
4
5  --·Change·the·Data·Type·of·salary
6  ALTER·TABLE·emp·ALTER·COLUMN·salary·TYPE·FLOAT8;
7
8
9
```

2.2.8. DDL - DROP - 1

00:30

You are working as a database administrator in a company, and you have been instructed to remove the `employees` table from the database. This table currently contains all employee information, and due to a change in the company's requirements, the table is no longer needed. Dropping the table will permanently delete the table along with all the data it holds.

Instructions:

- The employees table already exists in the database.
- Your task is to completely remove this table from the database.
- This action will permanently delete the table and all of its data, so it cannot be recovered unless a backup is available.

Write an SQL statement to drop the **employees** table from the database.

Expected Outcome:

- The **employees** table will be successfully removed from the database.
- All data within the table will be permanently deleted and cannot be recovered unless a backup is available.

Note: You are not required to create the table, The table is already available in the database.

Explorer

index.sql

```sql
-- Remove the table
DROP TABLE employees;

```

2.2.9. DDL - TRUNCATE - 1

00:28

You are a database administrator at a rapidly growing tech startup. The company's HR department has been maintaining detailed employee data in the `employees` table. However, due to a restructuring phase, the HR team needs to **temporarily remove all the records** from the table. The table's structure must remain intact so that new records can be added after the restructuring is complete.

The HR team has a tight deadline and needs this task to be completed without affecting any future operations that depend on the table structure, such as employee onboarding and payroll systems.

Requirements:

- **Objective:** Remove all records from the **employees** table, but keep the table's structure (columns, schema, constraints) intact.
- **Outcome:** The table must be cleared of data, but the column structure must remain unchanged so that new data can be inserted in the future.

`employees` table Schema:

Column Name	Data Type	Constraints
employeeID	INT	Primary Key
firstName	VARCHAR(50)	NOT NULL
lastName	VARCHAR(50)	NOT NULL
birthDate	DATE	-
hireDate	DATE	-
salary	DECIMAL(10,2)	CHECK (salary >= 0)

Data Snapshot:

employeeID	firstName	lastName	birthDate	hireDate	salary
			1980-01-15	2005-06-01	50000.00

Explorer

index.sql

```sql
-- Remove all records from the employees table
TRUNCATE TABLE employees;

```

2.2.10. DDL - 1

02:57

As the database administrator at a prestigious university, you are tasked with preparing the database for the new semester. The university needs to track student performance across various batches, and the system must store detailed information about each student, including their batch number, name, branch, and marks in three subjects.

The registrar needs to:

- **Create a table** to store student details, including batch number, name, branch, and marks for the first two subjects.
- **Add a new column** for marks in the third subject, as the curriculum now requires tracking marks for three subjects per student.
- **Clear any existing data** in the table before entering fresh records for the new batch of students.

Task 1: Create the student table with the following columns:

- batch_no (type: INT)
- name (type: VARCHAR(15))
- branch (type: VARCHAR(15))
- mark1 (type: INT)
- mark2 (type: INT)

Task 2: Alter the table to add a new column **mark3** (type: INT) to store the marks of the third subject.

Task 3: Truncate the table to remove all existing data, ensuring that the table is ready to store fresh records for the new batch.

Column Description:

Column Name	Description
batch_no	Indicates the batch number which is the type of integer
name	Indicates the student name which is the type of varchar having a size of 15 characters
branch	Indicates the branch of the student which is the type of varchar having a size of 15 characters
	Indicates the marks in the first subject which is the type of

Explorer

user.sql

```sql
-- Create the student table
CREATE TABLE student(
    batch_no INT,
    name VARCHAR(15),
    branch VARCHAR(15),
    mark1 INT,
    mark2 INT
);

-- Add the mark3 column
ALTER TABLE student ADD COLUMN mark3 INT;

-- Insert data
INSERT INTO student (batch_no, name, branch, mark1, mark2, mark
VALUES
    (101, 'Alice', 'CSE', 85, 90, 88),
    (102, 'Bob', 'ECE', 78, 82, 80),
    (103, 'Charlie', 'ME', 92, 88, 85);
....
-- Remove all existing data
TRUNCATE TABLE student;

```

2.1.39. And + OR Clause Together

02:50

In PostgreSQL, combining AND and OR conditions in a WHERE clause allows for complex querying by filtering rows based on multiple, layered criteria. This approach is essential for precise data retrieval, especially when the required dataset must satisfy a combination of different conditions.

Command, Syntax, and Examples

Selecting Rows with a WHERE Clause using AND and OR Conditions
Syntax:

```
SELECT column1, column2, ... FROM table_name WHERE (condition1 AND condition2) OR condition3;
```

Example:
In CodeKraft's customers table, to select customers who are either from 'New York' and have joined after 2020, or from 'Los Angeles':

```
SELECT customer_id, name, city, join_date FROM customers WHERE (city = 'New York' AND join_date > '20
```

This command fetches the customer ID, name, city, and join date for customers who meet either of the specified sets of conditions.

Task:
In the CodeKraft database, you need to retrieve data from the orders table for orders that were either placed in 2021 by customer 102 or have a total amount greater than 1000. The orders table includes columns for order_id, customer_id, order_date, status, and total_amount.

Table Schema for orders

Column Name	Data Type	Description
order_id	SERIAL	Primary key for orders
customer_id	INT	ID of the customer placing the order
order_date	DATE	Date when the order was placed
status	VARCHAR(50)	Status of the order

- [x] `SELECT * FROM orders WHERE (customer_id = 102 AND order_date >= '2021-01-01' AND order_date <= '`
- [ ] `SELECT order_id, total_amount FROM orders WHERE customer_id = 102 AND (order_date >= '2021-01-01`
- [ ] `SELECT customer_id, order_date FROM orders WHERE (order_date >= '2021-01-01' OR customer_id = 10`
- [ ] `SELECT order_id, customer_id, status FROM orders WHERE (customer_id = 102 AND order_date >= '20`

2.1.40. SQL Exercise

03:41

You are a procurement manager for a retail chain, tasked with analyzing inventory to identify products that meet specific criteria for restocking or promotional discounts. Your criteria are as follows:

- For **Electronics**: Select items priced below $150 to consider for a promotional discount campaign.
- For **Furniture**: Identify items with a stock quantity greater than 20 to evaluate their demand and decide if restocking is needed.

Your goal is to generate a detailed report of all relevant products meeting the above conditions.

Table Schema for products:

Column Name	Data Type	Constraints
product_id	SERIAL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL
category	VARCHAR(255)	NOT NULL
quantity	INTEGER	NOT NULL
price	DECIMAL	NOT NULL

Data Snapshot for products:

product_id	name	category	quantity	price
1	Wireless Mouse	Electronics	100	19.99
2	HDMI Cable	Electronics	50	9.99
3	Leather Sofa	Furniture	15	599.99
4	USB Flash Drive	Electronics	75	14.99
5	Office Desk	Furniture	25	199.99
6	Gaming Console	Electronics	20	299.99

Explorer

user.sql

SUBMIT

```
1  SELECT * FROM products WHERE (category = 'Electronics' AND price<150) OR
   (category = 'Furniture' AND quantity>20);
```

2.1.41. NOT Operator

01:41

Using a NOT operator in a PostgreSQL SELECT statement is pivotal for excluding specific data based on a condition. This operation allows for selecting rows where the specified condition does not hold true, offering a way to inversely filter data. It's particularly useful in scenarios where you need to exclude a subset of data that meets certain criteria.

Command, Syntax, and Examples

Selecting rows with a NOT clause
Syntax:

```
SELECT column1, column2, ... FROM table_name WHERE NOT condition;
```

Example:
In CodeKraft's customers table, to select customers who are not from 'New York':

```
SELECT customer_id, name, city FROM customers WHERE NOT city = 'New York';
```

This command fetches the customer ID, name, and city for all customers who do not live in 'New York'.

Task:
In the CodeKraft database, you are required to retrieve data from the **products** table for all products that are not in the 'Electronics' category. The products table includes columns for product_id, name, price, and category.

Table Schema for products

Column Name	Data Type	Description
product_id	SERIAL	Primary key for products
name	VARCHAR(255)	Name of the product
price	NUMERIC(10,2)	Price of the product
category	VARCHAR(100)	Category of the product

```
SELECT * FROM products WHERE NOT category = 'Electronics';
```

```
SELECT * FROM products WHERE category <> 'Electronics';
```

```
SELECT * FROM products WHERE category != 'Electronics';
```